Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 430808, 25 pages
doi:10.1155/2011/430808

Research Article

Sustainable Modular Adaptive Redundancy Technique Emphasizing Partial Reconfiguration for Reduced Power Consumption

R. Al-Haddad, R. Oreifej, R. A. Ashraf, and R. F. DeMara

Department of Electrical Engineering and Computer Science, University of Central Florida, P.O. Box 2362, Orlando, FL 32816-2362, USA

Correspondence should be addressed to R. Al-Haddad, rawadh@cs.ucf.edu

Received 16 January 2011; Revised 10 May 2011; Accepted 7 June 2011

Academic Editor: Scott Hauck

As reconfigurable devices' capacities and the complexity of applications that use them increase, the need for *self-reliance* of deployed systems becomes increasingly prominent. Organic computing paradigms have been proposed for fault-tolerant systems because they promote behaviors that allow complex digital systems to adapt and survive in demanding environments. In this paper, we develop a *sustainable modular adaptive redundancy technique (SMART)* composed of a two-layered organic system. The hardware layer is implemented on a Xilinx *Virtex-4* Field Programmable Gate Array (FPGA) to provide self-repair using a novel approach called *reconfigurable adaptive redundancy system (RARS)*. The software layer supervises the organic activities on the FPGA and extends the self-healing capabilities through application-independent, intrinsic, and evolutionary repair techniques that leverage the benefits of dynamic partial reconfiguration (PR). SMART was evaluated using a Sobel edge-detection application and was shown to tolerate stressful sequences of injected transient and permanent faults while reducing dynamic power consumption by 30% compared to conventional *triple modular redundancy (TMR)* techniques, with nominal impact on the fault-tolerance capabilities. Moreover, PR is employed to keep the system on line while under repair and also to reduce repair time. Experiments have shown a 27.48% decrease in repair time when PR is employed compared to the full bitstream configuration case.

1. Introduction

Current high-performance processing systems frequently consist of heterogeneous subsystems that depend on one another in nontrivial ways. Each subsystem is itself a multicomponent system with diverse capabilities. The organization of these subsystems is typically static; it is determined with great care at design time and optimized for a particular mode of operation. This design strategy is appropriate for systems that are accessible for repair when their components fail. However, systems that are unreachable once deployed present a different set of challenges. In these systems, the failure of a single component may result in large-scale inefficiency or even complete mission failure.

Therefore, electronic systems operating in demanding environments require increased capability for autonomous fault tolerance and self-adaptation, especially as system complexities and interdependencies increase. Hence, the goal of Organic Computing (OC) techniques [1, 2] is to create systems capable of adaptive and fault-tolerant behaviors. The OC paradigm is compatible with biologically-inspired computing concepts that emphasize the so-called "self-x properties" which emerge at the system level and represent life-like properties such as self-configuration, self-organization, and self-healing [2, 3]. These properties must be autonomous yet also must be sufficiently constrained to avoid the emergence of undesirable behaviors.

The OC paradigm is seldom tied to a particular platform or implementation, which makes it relatively broad in its impact and not restricted to any specific research or industrial context. Nonetheless, the immense flexibility of reconfigurable hardware devices makes them especially suited to hosting OC applications [4]. In particular, the fact that SRAM-based field programmable gate array (FPGA devices) can be dynamically reconfigured has made them a popular hardware platform for numerous OC systems [4, 5].

External environmental demands or internally driven performance demands may require a change in the configuration of a multicomponent system to maintain functionality and throughput throughout an extended mission [6]. For instance, a fault may occur in an individual component, which must then be replaced, refurbished to some degree, or otherwise bypassed. Although one could hypothesize that routine hardware failures would be a likely trigger for a configuration change, other mission-level considerations, such as a storage device reaching its capacity or the environment deviating from expectation, could be handled similarly. In either case, existing modules must be reconfigured; SRAM-based FPGA devices facilitate this flexibility by enabling dynamic device reconfiguration.

However, using FPGA devices rather than their application-specific integrated circuit (ASIC) counterparts in mission-critical applications is a double-edged sword. On the one hand, they allow the support of *self-x* capabilities through reconfiguration, but, on the other hand, such capabilities can introduce new fault vulnerabilities to the hardware. Transient faults, which commonly occur as single event upsets (SEUs) [7], are a primary source of concern when deploying SRAM-based devices in mission-critical applications, such as space applications [8]. SEUs can occur when a charged particle impacts the silicon substrate with enough energy to incur either a transient pulse in a combinational logic component or a state flip in a sequential component. The former is only articulated if a state component, such as a flip-flop, is affected by the transient signal. Hence, the effect of an SEU on combinational logic in ASICs can vanish without any repairs. However, SEUs hitting memory cells are more likely to cause damage because they flip the state of a stored bit, which affects the system until the relevant flip-flop is loaded with a new valid value. In SRAM-based FPGAs, in which even combinational logic is implemented using SRAM lookup tables (LUTs), SEUs gain amplified importance because every SEU is a state-flip that can affect both the sequential and the combinational logic. To this end, space-qualified versions of SRAM-based FPGAs, such as Xilinx's QPro [6], are commercially available for mitigating SEUs at the circuit level. Indeed, a new field of research that targets fault tolerance in reconfigurable platforms has emerged to take advantage of the inherent reconfigurability of FPGA devices. In conjunction with the use of high-reliability components, mission-critical applications can escalate the benefits of reconfigurability by employing fault-tolerance schemes to survive the various sources of failures that might affect reconfigurable resources.

In this paper, we present the design and implementation of SMART, which is a two-layered sustainable autonomic architecture for fault handling. The autonomous hardware layer is implemented on a Virtex-4 Xilinx FPGA device, while the software layer is intended to be on a PowerPC embedded core with internal configuration access port (ICAP) interface to the FPGA device to download configuration bitstreams (CBSs) for repair purposes. In this paper, in order to facilitate testing and verification, the software layer resides on a host PC that is connected to the FPGA via a Xilinx parallel cable IV. SMART is inspired by the OC paradigm, and thus the emergence of *self-x* properties is observed at the system level after assembling the individual parts into a single, integrated, fault-tolerant system.

The hardware layer implements a decentralized observer/controller processing loop to adjust the configuration of the system based on real-time mission information. It accomplishes this task using *RARS* [9], which is a general-purpose redundancy scheme that does not have a predetermined number of redundant modules like other fixed redundancy techniques commonly found in the literature (e.g., Duplex, TMR, and pair-and-spare) [10]. Instead, RARS can reorganize its components at run-time to provide the appropriate level of redundancy to match the mission status and requirements. The distributed controller function in RARS, which is called the *autonomic element (AE)*, monitors the status of the redundant parts that implement the user application, called the *functional elements (FEs)*, and collects the reports from various sensors to decide which configuration to select.

RARS is a power-conservative adaptive redundancy architecture that is only reconfigured to a high-power consuming design when multiple instances of the user application are needed to identify, mask, or repair faults. Other approaches, such as TMR, run in triplex mode even when faults are not present, consuming three times the dynamic power of the simplex configuration only to provide fault tolerance during brief intervals of the mission lifetime during which the system is subject to faults. RARS power benefits will be shown analytically and experimentally in the results section.

The fault tolerance of RARS is still restricted by the limited capacity of the available hardware to support alternative routing and logic for faulty parts. Therefore, a software monitoring and refurbishment layer that resides above the hardware layer is developed to provide active repair in the event of faults, either via scrubbing [11, 12], which involves rewriting the configuration memory with a fault-free CBS to correct any SEU occurrences in the configuration logic, or via a dynamic refurbishment of permanent faults using evolvable hardware (EHW) approaches [13]. The evolutionary approach employed in this work is a novel genetic algorithm (GA) that implements design practices for the organic nature of the system and thus is referred to as an organic GA (OGA). The software layer reads the performance and status of RARS and triggers the refurbishment procedure whenever the redundancy degree of RARS is not adequate to mask the faults.

Dynamic PR is adopted to improve the organic repair and the availability of the system. It significantly reduces the configuration time compared to the full bitstream configuration approach due to the small size of the bitstream. In addition, it allows the system to remain online while its faulty parts are being reconfigured; this helps increase the availability of the system by enabling it to maintain functionality even during repair. Dynamic PR is exploited during the scrubbing phase of the repair cycle in addition to the OGA fitness evaluation stage to reconfigure candidate solutions on the FPGA for intrinsic fitness assessment.

To illustrate the organic capabilities of SMART, the well-known Sobel edge-detection [14] application was implemented on the FPGA as a real-life case study. After combining all hardware and software modules into one integrated platform, the system's behavior was scrutinized while processing a real-time video stream under various fault scenarios. The hardware layer demonstrated the emergence of self-monitoring and self-organization properties that allowed the system to sustain performance even in the presence of successive faults. When the number of faults exceeded the capabilities of the hardware layer, the higher-level software layer augmented the response through self-configuration and self-healing.

Figure 1 depicts the high-level view of SMART repair methods and the various events that trigger their execution. The central state of SMART operation is the fault-free operation (1) that requires only RARS's self-monitoring techniques to detect the occurrence of faults. An SEU can impact the FPGA resources and cause a single bit flip in one of the LUTs. This LUT may fall either on the data path of the application, that is, a user register that stores an intermediate calculation value, or on the logic path, that is, an LUT that is programmed to implement the intended circuit functionality. SEUs that affect flip-flops in the user logic can be overwritten by subsequent operations without any repair intervention. This type of fault is classified as transient and normally fades away in the regular execution cycle. The transient effect can be masked with redundancy techniques (2) until the fault is corrected. In fact, it is disadvantageous for the system to trigger repair based on transient data path fault indications, which can be avoided by using a watchdog timer to determine the persistence of faults.

However, if the soft fault affects an LUT in the reconfigurable logic, then the bit flip will remain intact until the unlikely event of another SEU impacting the exact same location. A bit flip in the logic path can be more harmful to the application because it changes the truth-table content of the affected LUT and thus alters the behavior of the circuit. This type of SEU cannot be ameliorated in subsequent operations because the affected element is not written by the user application; thus, it must be explicitly rewritten by reloading the correct bitstream via scrubbing (3).

Next, consider if radiation leads to pathways for electromigration and accelerated aging effects [11]. This type of local permanent damage (LPD) can be modeled as a stuck-at fault at one of the LUT inputs. Unfortunately, scrubbing techniques that rewrite the CBS contents will at best give up after a number of retries or at worst may usurp the mission, taking the device off line to repeatedly attempt to overwrite a permanent fault. In that case, a permanent fault handling technique is required to circumvent the stuck-at faulty resource and thus repair the user application.

The self-configuration of spares (4) tries to avoid the faulty resource by consecutively reconfiguring the faulty FE with design-time preseeded bitfiles, each of which exclusively avoids a set of LUTs in the physical FE area. By doing so, SMART searches the set of spares for one spare that can hide the fault by simply not using the damaged LUT. Carrying spares is a common technique for fault tolerance due to its simplicity and quickness; it is limited, however, to the number of carried spares and cannot really adapt at run-time to handle fault scenarios that were not considered at design time when the spares were designed.

As a remedy, SMART added one last-resort repair mechanism that is invoked when all other techniques fail to repair faults. This technique is the evolutionary OGA (5) repair that is not restricted by the number of spares or any other design-time consideration. Instead, it can heuristically search for alternative circuits that can bypass the faulty resource and yet produce the expected output. Such technique can be slow and unpredictable, but the fact that it is delayed to the very end of the repair cycle makes it an acceptable alternative to accepting degraded level of operation of the application.

2. Related Work

SMART aims to achieve autonomous fault tolerance by employing OC concepts. Therefore, this section explores fault-tolerance OC efforts in the literature. In addition, we present literature survey for some of the most prominent fault-tolerance techniques in the literature and compare them against SMART. More details about fault-recovery techniques in FPGA-based applications can be found in [6].

Related works have explored techniques useful for the development of OC systems from various theoretical and practical perspectives. A frequent focus shared across researchers has been the design of OC architectures and development methodologies for systems with the potential to exhibit increased reliability and sustainability.

For example, the objective in [18] is to introduce an OC-inspired design of a reliable system-on-a-chip (SoC) to provide broader coverage of faults than classical design methods can offer. The platform is demonstrated on five-stage RISC pipeline architecture with global error counters to detect transient errors. A customized microrollback technique is used to correct errors with a delay of two cycles in the example RISC pipeline. This OC system is shown to possess self-calibration and self-healing properties when affected by transient errors, with specific fault-tolerance techniques tailored for CPU pipelines. SMART expands on this by presenting a generic cross-platform fault-tolerance technique that can handle both transient and permanent faults.

In [19], an observer/controller architecture was developed to provide a generic template to design control architectures for OC systems without extension to a hardware prototype implementation. This organic framework mainly targeted self-organization in a simulated environment and recommended thorough empirical studies of OC systems in different domains due to their flexibility and autonomous nature. SMART employs similar observer/controller architecture and demonstrates the architectures in a real edge-detection use case running in error-prone environments.

In [20], digital on-demand computing organism (DoDOrg) targeting real-time systems is presented. The system model is based on biological principles to achieve the desired self-x properties; it is divided into processing cells representing human cell analogs, middleware control representing organ analogs, and high-level control representing a brain analog. The work presents an approach to organic

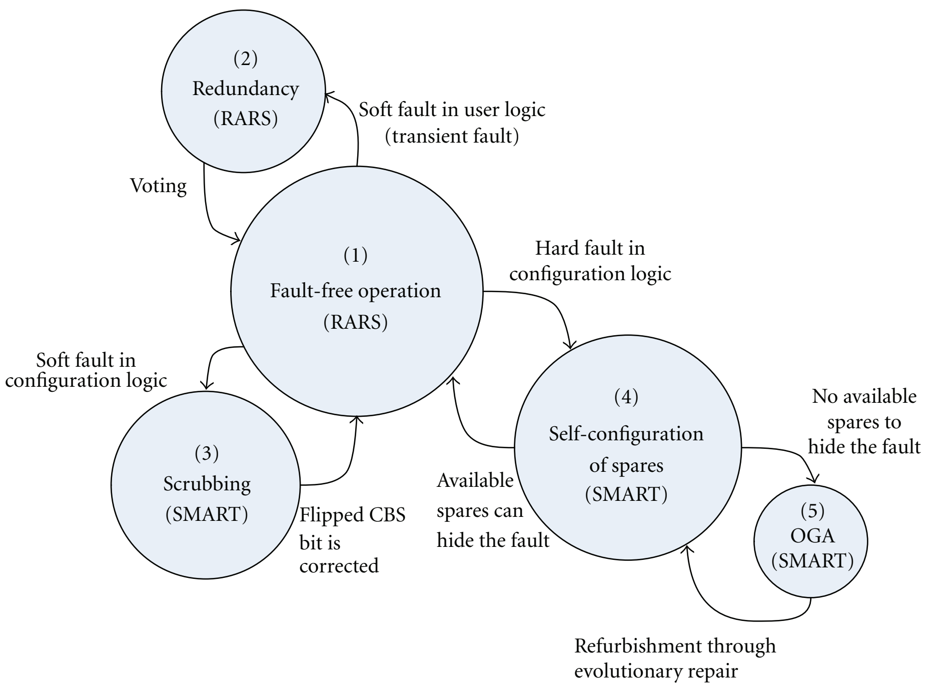


Figure 1: High-level View of SMART's Repair Methods.

computing that shows many of its desired self-x properties along with power management. While the viability of this system is shown in a simulated environment, the transfer to a real robot system is sought in a later phase.

In an attempt to practically realize DoDOrg on FPGAs, a framework to achieve a decentralized configuration and power management scheme is shown in [4]. This work considers FPGAs as the most viable computing platform for OC systems due to the enormous benefits of reconfigurability. However, the work identifies the centralized nature of the FPGA ICAP as the main contradiction to the crucial decentralized requirement of OC systems. Therefore, a platform in which each computing node can autonomously and independently request its reconfiguration through the ICAP is presented. Similarly, power consumption is managed by individual nodes at run-time to attain the desired virtual decentralization of the ICAP.

As for the generic FPGA fault-tolerance methods, one of the most common techniques for mitigating unwanted configuration memory changes is scrubbing [12]. Scrubbing involves overwriting of the configuration memory at periodic intervals with a configuration that is known to be fault-free. Moreover, this process can be augmented by reading back the configuration memory and comparing it with a configuration that is known to be good to isolate the erroneous frame(s) so that they can be rewritten using PR. Scrubbing techniques fail when the stored configuration is damaged or when the fault is caused by permanent hardware resource failures, in which case more elaborate repair techniques targeting permanent faults should be employed, such as the evolutionary repair algorithm presented in [21] and in this work.

Table 1 presents a comparison between SMART and other techniques. All surveyed techniques, except conventional TMR, employ some form of fault recovery mechanism to restore the original fault-free system status. TMR is a passive technique which employs spatial voting to mask the faults. The area and power overhead for the TMR approach is three times the area associated with a single module (O_{FE}) plus the overhead associated with the voting logic (O_V).

In [13], an offline genetic algorithm refurbishment technique to handle hard faults is presented. All the modules are simulated with faults representing a worst-case scenario, and the evolution-based refurbishment is performed on all three modules for recovery. The overhead associated with the GA-based repair is represented as O_{GA}. This cost can be used to include all GA-based control mechanisms and the spare resource allocated for GA-based refurbishment.

Reference [15] on the other hand presents a technique based on design-time allocation of fine-grain spares at the CLB level. One CLB is allocated as spare for a design-time defined group of CLBs, and multiple configurations are generated such that one fault can be tolerated in such a group. Area overhead on average for the chosen benchmarks is reported as 5.4%, which is considerably less than the TMR. This scheme, however, does not include any fault detection mechanism.

STARS [16] employs run-time built-in self testing (BIST) by roving across the FPGA fabric. This technique covers fault detection, isolation, and repair with minimal application area overhead. But the time to detect a fault can be quiet high and as much as 8.5 M erroneous outputs may be produced before being able to detect the fault [22]. Further, the fault detection process employs continuous reconfiguration and thus incurs huge power overhead.

The Jiggling approach [11] employs spatial TMR for masking the fault and a (1+1) evolutionary strategy to

Table 1: Comparison between SMART and other prominent fault-tolerance approaches.

Approach	Fault handling method	Fault detection		Resource coverage		Fault isolation granularity	Power overhead area cost
		Latency	Hard faults	Logic	Comparator		
TMR	Spatial voting	Negligible	No	Yes	No	Voting element	$3 * O_{FE} + O_V$
Vigander [13]	Spatial voting and offline evolutionary refurbishment	Negligible	No	Yes	No	Voting element	$3 * O_{FE} + O_V + O_{GA}$
Lach [15]	Design-time fine grain redundancy-based reconfiguration	Not addressed	No	Yes	Not addressed	Group of predefined CLBs	Fault detection mechanism is not addressed
STARS [16]	Online BIST	Depends on geometry of device	Yes	Yes	Yes	Single LUT	O_{FE} + reconfiguration controller
Garvie [11]	Spatial voting and online (1+1) ES	Negligible	Yes	Yes	No	Voting element	$3 * O_{FE} + O_V + O_{GA}$
Keymeulen [17]	Design-time population-based fault insensitive designs	Not addressed	No	Yes	Not addressed	Not addressed	Fault detection mechanism is not addressed
SMART	Adaptive redundancy, diversity-based configurations, OGA	Negligible	Yes	Yes	No	Voting element	Analyzed in Section 6.4 Power Savings

refurbish the identified faulty module. The power and area overhead of this technique can be essentially considered same as that of TMR. The work concludes that hard-fault tolerance is essential for fault tolerance of FPGA devices in harsh-environment deployments.

An evolutionary-based method [17] is introduced to generate a population of individuals at design time that are resilient to a set of predetermined type of faults according to the planned mission. This design-time process is tested by employing the design-time generated configurations to overcome the expected fault pattern at run-time. This scheme is classified as static, due to its inability to accommodate all possible faults at design-time.

Finally, software control in autonomous applications is an essential part of the overall fault-tolerance package. In [23], a multilayer runtime reconfiguration architecture (MRRA) framework capable of communicating with the FPGA through high-level API calls is introduced. This modular architecture has a hierarchical framework that supports different functionalities at an abstract level because each functional layer can do its job independently of other working layers. In an extension of MRRA, an intrinsic evolution platform is implemented [21] by introducing genetic algorithm operators at the logic layer to achieve successful design and repair of digital circuits on Virtex II Pro FPGA. In this paper, we use the same intrinsic evolution platform and extend the direct bitstream manipulation to Xilinx Virtex 4 FPGA devices.

3. Organic System Architecture

Figure 2 depicts the detailed architecture of the lab prototype of SMART. The software-based repair layer is implemented on a host PC to aid in experiments and validation. The deployment system is intended to have the software components implemented in an embedded PowerPC processor that comes with many commercially available Xilinx FPGA boards.

The lower half of the figure shows the organic hardware layer where the system can accommodate one or more FPGA boards, each of which has one or more RARS modules.

In the experimental setup, the two layers are connected via a Xilinx Parallel Cable that connects between a standard Joint Test Action Group (JTAG) [21] port on the FPGA and the parallel port on the host PC. On the FPGA, the JTAG communicates with RARS via the General-Purpose Native JTAG Tester (GNAT) [21] platform. The messages themselves are communicated using a communication protocol that was designed specifically for this system. This communication link carries messages between the two layers as part of the fault-tolerance algorithm and also transmits the CBS to reconfigure parts of the system as needed. The intended deployment platform should implement the software layer on an embedded PowerPC and utilize the ICAP interface for device configurations.

The software layer communicates with the hardware through a multithreaded communication manager, which is responsible for abstracting all hardware complexities and providing messages to the various software components. These components include the human interface module (HIM), which converts the binary message into human-readable text files and vice versa; this extension was crucial to validate and debug thousands of binary messages that were transmitted between the two layers during the prototyping phase. The software layer also includes the scrubber and the OGA repair modules. The hardware and software layers depicted in Figure 2 will be described in the next two sections.

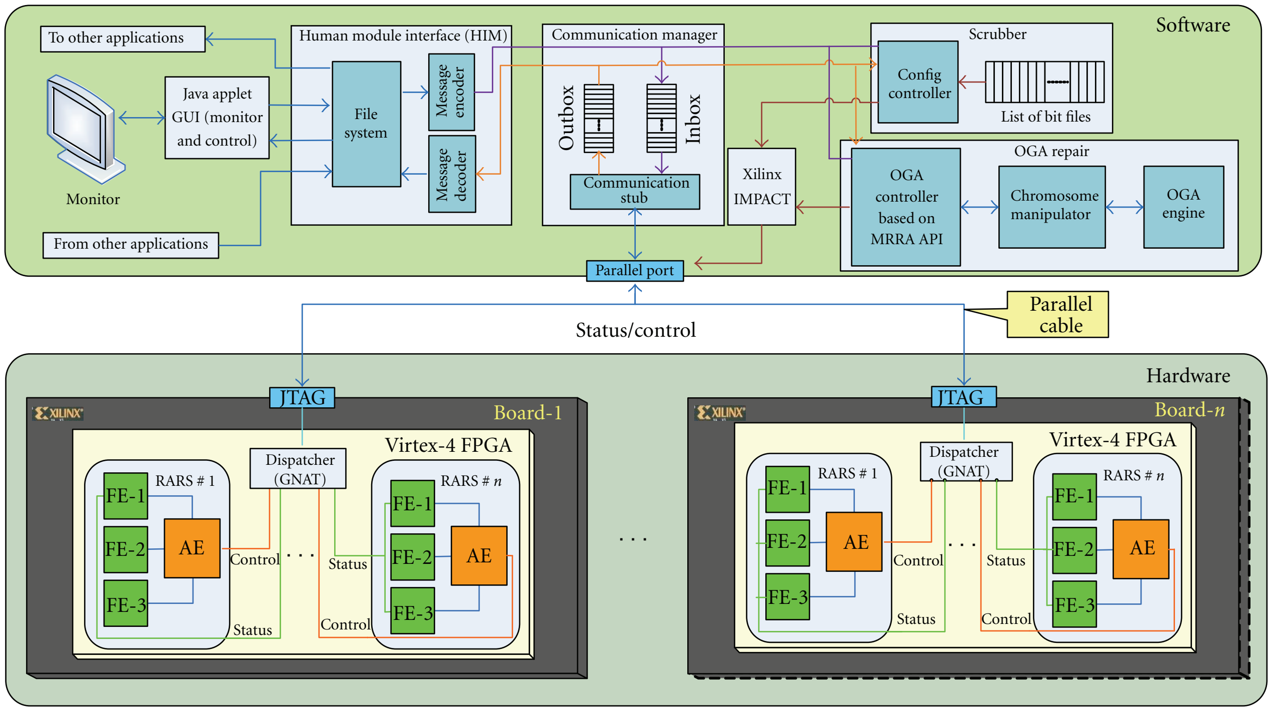


Figure 2: SMART prototype top-level hardware and software architecture.

3.1. Hardware Layer. The hardware layer consists of one or more RARSs and dispatchers configured on one or more FPGA boards. The RARS module comprises the smallest integrated unit in the hardware platform; it consists of one AE and three identical FEs. The AE is application independent; it contains the logic that drives the organic behavior by actively reorganizing the available FEs. However, the FEs represent the application dependent user implementation of the desired functionality. Therefore, the FEs are the only modules that need to be modified for the system to support new applications.

Having three FEs in each RARS module illustrates the common practice of employing a TMR configuration in redundancy-based fault-tolerant systems. There is no loss of generality that prevents RARS from accommodating $2n + 1$ FEs for any $n > 0$.

One straightforward approach for RARS is to initially enable two FEs while the third is kept off line as a cold spare. Upon finding a discrepancy between the two outputs in the duplex mode, the AE switches to the TMR mode of operation by placing the stand by third FE on line and activating a voting scheme among the three FEs to mask any single fault. While the duplex mode has the shortcoming of expending clock cycles from the instant it detects a fault until the correct functional output is regained, it reduces the required dynamic power compared to a conventional TMR in the no-fault scenario. Moreover, the fact that the stand by FE is normally off line makes its resources available for use for any other purposes.

3.1.1. RARS Motivation as a Hybrid of Approaches. Traditional reliability techniques often rely on the concept of redundancy. Redundancy is the addition of resources beyond what is actually needed for normal system operation to retain functionality when faults occur. *TMR* requires three functionally identical modules that perform the same task in parallel and a voter that outputs the majority vote of the three modules [24]. *Concurrent error detection (CED)* [25] approaches rely on a duplex configuration and discrepancy detection among the output bits of the redundant modules. Both TMR and CED can increase reliability using *stand-by sparing* approaches, whereby *hot spares* are kept in an idle state and thus are ready to be called into action once required. *Cold spares*, in contrast, are kept shut down and thus do not consume power but incur delay before they are able to replace faulty modules.

A tradeoff arises between the increased system dependability and the overhead associated with having redundant parts. For instance, duplex systems maintain one redundant element but cannot mask faults in real time. Adding one module to a duplex configuration enables it to mask faults via TMR techniques at the expense of extra area, power, and cost. This compromise is usually hard to decide at design time.

In addition, mission-critical applications are impacted by many parameters, some of which can only be decided at run-time. For example, an edge-detector circuit is of extreme importance when it is operating on a critical video stream, such as a moving object in a surveillance recording. In such cases, it is usually necessary to quickly mask any faults that might occur because any loss of detection capabilities is intolerable and can affect the overall mission objectives. However, if the same edge detector is operating on a still scene in a surveillance recording, then it might be possible for the system to tolerate some degradation in the output

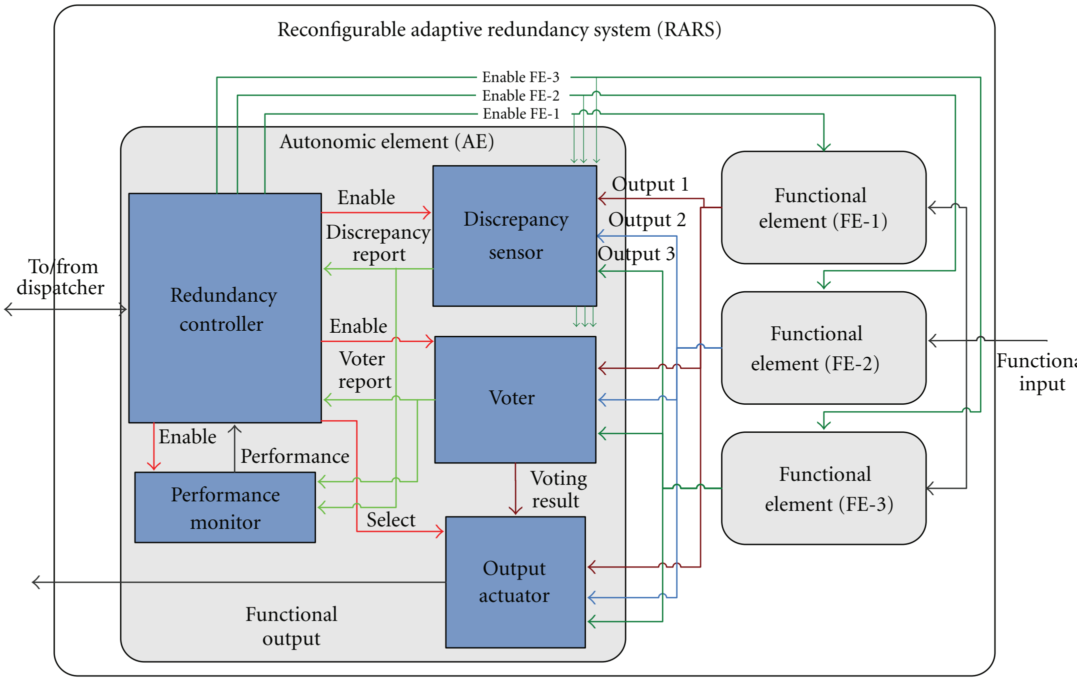


Figure 3: Reconfigurable adaptive redundancy system (RARS).

because the generated image can still be analyzed later or simply omitted due to the lack of action in the scene. TMR may be a wise choice in the former case, whereas a duplex configuration might be a better option in the latter. This scenario is an example of a system that shows changing reliability needs at different mission stages.

Whereas many other studies have decided the redundancy levels in their systems at design time [8, 26, 27], we sought an adaptive solution by deferring the decision regarding which level of redundancy to support until run-time. Thus, the choice can be enhanced by mission information and status to make it an efficient compromise between the desired reliability and the associated overhead in terms of area and power.

3.1.2. RARS Architecture. The proposed RARS architecture is shown in Figure 3. The functional input is delivered directly to the three FEs for evaluation. The outputs of the FEs are then sent to the AE to be processed by the following five modules.

(1) Discrepancy Sensor (DS). This component uses the three FE outputs to detect discrepancies between any pair of enabled FEs. This module is only activated when RARS is running in the duplex mode; otherwise, it is disabled to save power.

(2) Voter. The voter module performs bitwise voting among the three FE outputs and produces the majority vote. It also generates a report that conveys any of the condition codes listed in Table 2. The voter is enabled only in the TMR mode and otherwise is disabled to save power.

Table 2: Possible values for the voter report.

Voter report	Description
000	No discrepancy among the three FEs
001	FE1 is discrepant from the other two FEs
010	FE2 is discrepant from the other two FEs
100	FE3 is discrepant from the other two FEs
111	All FEs are discrepant (m-bit, $m > 1$)
101	Voter is disabled

(3) Output Actuator (OA). This module performs a 4×1 multiplexer function. The inputs come from the outputs of FE1, FE2, FE3, and the voter. The output drives the overall system's functional output. This module signifies the flexibility of the AE compared to other fixed-redundancy techniques because the AE can select from all of the simplex configurations in addition to the majority vote output.

(4) Performance Monitor (PM). This module samples the DS and the voter report to provide reports that reflect the aggregate performance of the system. The PM is periodically polled by the software layer during repairs to acquire system performance to convey the fitness value of the evaluated individuals.

(5) Redundancy Controller (RC). This is the core element in the AE; it is responsible for the unit awareness and for sending status reports and receiving control signals to or from the software layer. In SMART, the RC is a finite state machine (FSM) that encodes all possible system configurations. The inputs to this state machine are the reports from

the various modules, such as the DS, the voter, and the PM. The output drives the "Enable" signals for all the modules and the selection lines for the OA. Moreover, this module contains the communication logic of the dispatcher and the input and output buffers that store the incoming and outgoing messages.

3.1.3. Possible Configurations. To obtain adaptive levels of redundancy, RARS uses real-time performance feedback based on the mission objectives to dynamically reorganize its modules into one of the following configurations.

(1) Simplex. The RC disables two FEs, the DS, and the voter. The OA propagates the enabled FE output. This configuration allows the highest power conservation if that is a priority. It is also practical during noncritical stages of missions. The simplex configuration can also be enabled during repair in a pair-and-spare scheme.

(2) Duplex. The DS is enabled to inform the RC in the event of output disagreement between the two enabled FEs. The OA is set to one of the enabled FEs. This configuration is only used for applications that can tolerate temporary degradation in output quality until the RC takes further repair action. The system can run in duplex mode while repairing a faulty module to detect additional faults in the online modules.

(3) TMR. The voter and all FEs are enabled, and the OA propagates the voter output. Only the DS can be disabled, as the voter report is able to convey all needed information. The system can maintain 100% correct throughput in the TMR mode even if one module is faulty. Even with the existence of multiple faults, design diversity and compensating module faults [10] can still assist in generating a correct vote. The TMR configuration is utilized in this platform when the system is repairing a faulty FE because PR will allow the system to maintain full functionality while the FE is repaired.

(4) Hybrid Mode. Many temporal configurations can be supported by RARS. For example, an application can run in simplex mode but switch to duplex periodically to detect discrepancies. Another usage example might be an application that has a duplex reliability requirement except during certain stages of the mission, during which it can switch to TMR to meet reliability needs. Downgrading is also possible based on reliability needs, and the arrangement of FEs can be dynamically reconfigured back to the original configuration once the operating behavior has changed accordingly.

3.2. Software Layer. The software layer controls the higher-level throughput of the system by monitoring performance and enabling active repair when the performance dips below an acceptable level, as specified by the mission requirements. The software layer serves two main purposes that are described below.

The first purpose is to provide an interface to monitor and control the hardware, if needed. To that end, a Java applet GUI has been created to depict the hardware status schematically and show the status of each component. The applet user interface [28] shows the following information.

(1) FE status: online, offline, faulty, fault-free, or under repair.

(2) AE configuration: simplex, duplex, or TMR.

(3) Performance level: the number of reported discrepancies divided by the total number of evaluations.

(4) Log of the transmitted messages: The messages are recorded as a paper-trail of hardware status changes.

The second purpose of the software layer is to enable higher-level autonomous recovery techniques. First, the scrubbing technique maintains functionally equivalent, physically distinct configurations and repetitively reconfigures the faulty FE until the faulty element is excluded from the logic path. Second, we have demonstrated in our experimental work that SMART is able to recover simulated hard faults by means of OGA. The fitness function was set to be the instantaneous performance level of RARS over a recent window of inputs.

3.2.1. General Architecture. The top half of Figure 2 shows the architecture of the software layer. The communication manager (CM) is a multithreaded C++ module that acts as the parallel port driver to communicate messages with the hardware. The HIM converts the binary messages in the CM queues into human-readable messages that are stored in a predefined directory on the file system and vice versa. The message decoder consults the communication protocol opcode table and generates text files that represent the messages. For example, this decoded message illustrates the status of FE #2 in RARS #1 as being on line and fault-free:

```
MSG_NAME: FE_STATUS_REPORT
MSG_CODE: 3
AE_ID: 1
FE_ID: 2
STATUS: 1 (ONLINE AND FAULT-FREE)
```

This platform provides a bidirectional communication link between the organic hardware and any user application that needs to monitor and/or control it. Any application that complies with the protocol message format can communicate with the hardware layer by storing textual messages in a predefined directory on the host PC. The encoder periodically polls for the message files, encodes them into binary representation, and stores them in the inbox queue of the CM to be sent to the organic hardware.

The scrubber and the GA repair modules can be seen to the right of the CM in Figure 2. These modules are described in detail in the next two subsections. Both of them can reconfigure the FPGA by executing batch files that invoke the Xilinx iMPACT tool [29] to perform partial device reconfiguration using the parallel Cable IV.

3.2.2. Scrubbing and Design Diversity. The RARS-centric techniques are sufficient to recover from transient faults in the user logic. SEUs in the configuration logic and hard faults cannot be indefinitely masked by redundancy because any further faults can shift the voting results toward the faulty FEs. Thus, in such cases, RARS will signal to the software layer of SMART to intervene and help fixing this type of persistent faults.

SMART begins by assuming that the persistent fault is caused by an SEU in the configuration logic (soft fault) that caused the flipping of one or more LUT bits. SMART handles this via scrubbing the bitfile with another CBS to correct the impact of the SEU and thus restore the correct functional operation of the circuit. Scrubbing entails fetching the CBS from an off-chip ROM via PowerPC APIs, reconfiguring the faulty FE via the ICAP, reading back the freshly downloaded bitfile to compare it to the ROM-based golden image, and finally monitoring the discrepancy for a sufficient number of evaluations to ensure that the fault is indeed corrected by scrubbing.

If the fault is not corrected by simple CBS scrubbing, SMART concludes that it is caused by a hard fault that requires extra repair effort. It starts by dynamically reconfiguring a set of design-time generated spares that have different area constraints to guarantee the avoidance of each and every LUT in at least one of the spares. This will ensure that each faulty LUT can be avoided by, at least, one available spare.

The design-time spare generation is accomplished via the Xilinx PROHIBIT constraint in the Xilinx User Constraint File (UCF) [29]. The PROHIBIT constraint allows the designer to specify a set of LUTs that should be avoided during the placement stage of the bitfile creation process. For example, the following constraint will exclude all slices in the range between locations (0, 33) and (13, 33):

```
CONFIG PROHIBIT = SLICE_X0Y33:SLICE_X13Y33
```
(1)

The same HDL entry is used to generate multiple configuration bitfiles, each with different UCF settings that exclusively prohibit the use of a set of slices. When a fault occurs, the scrubber successively downloads the bitfiles to the FPGA and searches for a configuration that prohibits the use of the stuck-at slice, in which case the fault will be corrected throughout a window of readings. If none of the preseeded bitfiles was able to hide the erroneous output, perhaps because there is more than one faulty LUT in the FE that cannot be excluded by any spare, the scrubber ceases to be efficient and will consequently request the intervention of the OGA repair.

The scrubber is the first line of recovery from faults that cannot be handled by RARS reorganization techniques. SMART relies on lazy scrubbing [11] such that only the discrepant FE in a TMR configuration is partially reconfigured while the system remains on line; the other two fault-free FEs, along with the voter, guarantee that the system can maintain correct overall output while the faulty FE is being scrubbed. A tile-based reconfiguration approach for fault tolerance is covered in detail in [15].

3.2.3. Organic GA (OGA). Our autonomous fault-tolerance method incorporates GA-based repair as an integral part of the repair cycle because it offers hard-faults active repair that is independent of the carried spares. However, the GA is a nondeterministic process that can affect the flexibility of SMART if not designed efficiently. Thus, three properties that can enhance the efficiency of the GA for an organic system were addressed.

(1) Direct Bitstream Evolution. Genetic representation is the process of mapping from the visible traits of the application (i.e., phenotypes) to the genetic coding of the chromosomes (i.e., genotypes) and vice versa. The mapping from phenotype to genotype (PTG) is performed only once during the design stage of the GA, and it requires special care to capture the building blocks that the GA needs to evolve to realize the desired solutions [30]. The mapping from genotype to phenotype (GTP), however, is applied every time the individual fitness is evaluated to transform chromosomes into physical individuals that can be evaluated on the application platform.

This two-way mapping can be a source of errors and complications if the distance between the genetic encoding and the phenotypic realization is large. For instance, if the GA evolves the HDL code of the FE to repair its circuit realization on the FPGA, then every time the chromosome is evaluated, it must undergo synthesis, mapping, placement and routing (PAR), bitfile generation, and FPGA reconfiguration, which is a huge overhead to endure for every evaluation. For that reason, we designed OGA to use direct bitstream evolution, whereby the chromosome is selected to be the FPGA raw bitfile. This selection puts the burden of PTG on the FPGA vendors (Xilinx in this case) and abridges GTP to a mere Xilinx iMPACT invocation to download the CBS onto the FPGA.

Direct evolution of a bitstream depends on details about the LUT mapping between the CBS and the actual device. The encoding of the bitfile must be manipulated to be able to apply genetic operators such as crossover and mutation to the relevant sections of the long bit array. This overhead is still considered feasible given the vast advantages of direct CBS evolution in terms of increased performance and reduced mapping effort. To directly manipulate the CBS, it is necessary to decode its bits to understand how to locate and modify specific LUTs and thus change the behavior of the resulting circuit. To that end, we extended the Virtex-2 approach that we previously developed in [21] to perform direct bitstream evolution on Virtex-4 devices.

The CBS contains the LUT contents evolved by the GA in addition to other information such as routing, checksums, and header information including the device signature and the time of bitfile creation. As shown in Figure 4, we implemented an LUT mapping module to map the location of LUT_X_Y, where X and Y are valid coordinates inside the evolved FE, to the correct offset in the CBS file. This mapping is not completely documented in any of the Xilinx application notes; rather, it was discovered through repetitive trial-and-error experiments.

Each experiment was designed to discover the mapping between one of the LUT coordinates and the corresponding bit offset in the CBS file. This discovery was accomplished

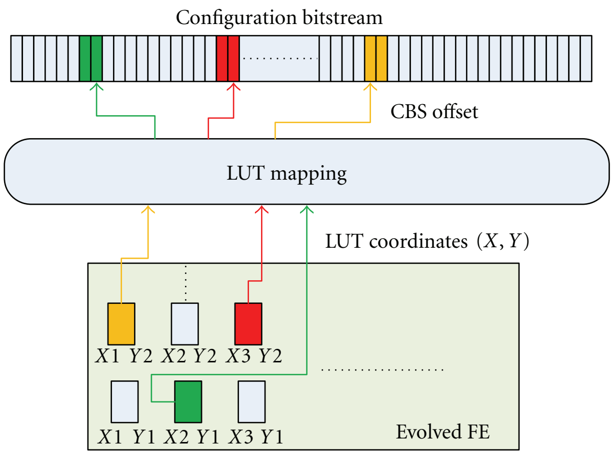


Figure 4: Mapping from LUT coordinates to CBS offset.

by viewing the native circuit description (NCD) file using the visual Xilinx FPGA editor tool [29] and negating the content of one known LUT. The NCD files before and after the negations were used to generate bitfiles with the same bit generation (bitgen) options [29]. The two resulting CBSs were then compared using a hex comparator. The 16-bit LUT content could be readily identified by monitoring the inverted bits between the two hex files; other differences resulting from the header and time stamps were usually located at the beginning of the bitfiles and thus promptly discarded. After many trials, a relation can be inferred between the XY of the LUTs and their offsets in the file, or rather store a lookup table that contains all of the used LUTs along with their corresponding offsets in the CBS file, to assist in the mapping.

(2) Intrinsic Fitness Evaluation. There are two methods to measure the fitness of the evolved individual. The common approach is extrinsic evaluation [13], which operates on a software model of the FPGA device. This abstraction simplifies the experiments and can be tuned more dynamically. However, the resulting representation has to undergo mapping and PAR on the target FPGA at deployment time. This step imposes a risk of incompatibility between the device's physical constraints and the software model that was used in simulation, thereby possibly leading to incorrect solutions. Instead, the OGA performs intrinsic fitness evaluation [31], whereby the hardware itself is used to measure the fitness of the evolved individuals. All of the device's physical constraints are considered during the process, and even the output is measured from the FPGA device while it processes the functional inputs of the user application.

Therefore, the system can remain on line during fitness evaluation provided that there are redundant parts to compensate for the evolved individual. Intrinsic evaluation requires that the evolved circuit be configured into the FPGA device each time the fitness is measured. This process is made feasible because of the direct bitstream feature of OGA, which means that the GTP requires only a Xilinx iMPACT device configuration to place the circuit on the FPGA.

(3) Model-Free Fitness Function. An accurate fitness function is a critical factor in designing an efficient GA because it determines the shape of the problem landscape that the GA will search [30]. This process can be extremely complicated in real-life engineering problems because it requires capturing all the attributes that distinguish a good solution from other ones. In addition, it is highly dependent on the problem domain because what is appropriate for one particular purpose does not usually fit other purposes.

Because SMART is intended to be a generic platform that fits any application domain, the OGA employs a novel, application-independent, model-free fitness function that can be ported to other applications with minimal effort. This was made possible because of the design of RARS, which enables run-time discrepancy detection between the evolved FE and other redundant, fault-free one(s). The model-free fitness function quantifies the fitness of the evaluated FE by counting the number of discrepancies between its output and other fault-free FEs. The number of discrepancies over a window of evaluations is stored in the PM and is reported by the RC to the OGA engine using a performance report message. This value quantifies the deviation between the evaluated individual's fitness and the ideal one, in which low values indicate fitter individuals. Therefore, the GA becomes a minimization optimizer for this value.

It is important to note that this model-free fitness function is only possible when the goal is to repair a faulty circuit and there is another redundant circuit on the FPGA that can produce the same functionality. This condition does not pose any limitation on redundancy-based fault-tolerant systems because the redundant parts are activated anyway to mask faults and thus maintain correct functional output. The OGA takes advantage of that activation and implements the model-free fitness function.

The OGA platform consists of the following modules [21].

(1) GA Engine. This is a C++ application that implements a customizable, standard GA (SGA). This module is platform independent; it encapsulates the implementation of the SGA, including the population data structures, the functionality for selection and replacement, and other standard GA operators such as mutation and crossover.

(2) Chromosome Manipulator. This is a C-based library that abstracts the underlying hardware from the perspective of the OGA engine. It provides hardware-independent abstraction of the genetic operators so that they can be executed with regard to the LUT boundaries in the long CBS string.

(3) Multilayer Run-Time Reconfiguration Architecture (MRRA). This is a set of APIs that facilitates communication with the target FPGA device [23]. This module handles direct bitstream manipulation and decoding and includes the LUT mapping module depicted in Figure 4.

(4) Bitstream File. This is the PR bitstream file that represents the FE design.

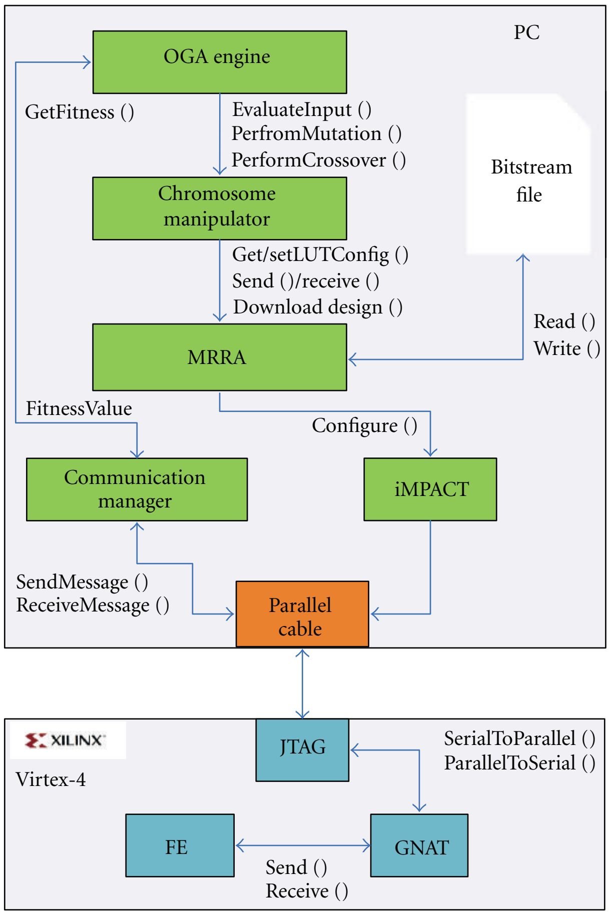


Figure 5: Intrinsic evolution platform.

Figure 5 shows the complete experimental OGA platform. The OGA engine relies on the chromosome manipulator to perform platform-independent mutation and crossover operations. It also reads the fitness values from the communication manager, which in turn acquires them directly from the hardware via the communication protocol messages. The MRRA module operates directly on the bitstream using the LUT mapping module shown in Figure 4 and then execute a batch file that runs the iMPACT tool, which performs boundary-scan device-chain initialization and then programs the chip. All communication proceeds via the parallel port from the host PC side and the JTAG port from the FPGA side.

The OGA creates the initial population based on the partial bitfile that was used to configure the original faulty FE. It generates a copy of the bitfile for each individual in the initial population and then randomizes its LUT bits to promote genetic diversity that could lead to more innovative solutions.

Next, each individual's fitness is evaluated intrinsically by downloading its bitfile to the FPGA using iMPACT. The OGA engine then requests the fitness value of the evaluated individual using a PERFORMANCE_REQUEST message that is sent to RARS through the JTAG-GNAT interface. The RC reads the PM counters, which are updated periodically based on the actual run-time functional inputs that the FEs are processing. It then formulates PERFORMANCE_REPORT as a reply message and sends it back to the GA engine.

After evaluating the fitness of all individuals, the OGA selects the individuals that will participate in the creation of the next generation using a tournament selection of size 2. This value was set based on preliminary runs designed to locate the most promising GA parameters for the experimental work. The selected individuals are then mated to create the offspring using single-point crossover and conventional bit-flip mutation operations. Both operators are executed on the raw bitfile, as mandated by the direct bitstream evolution premise. The mapping between the LUT coordinates and its actual location in the bitfile is abstracted using the LUT mapping module demonstrated in Figure 4 to map LUT_X_Y coordinates to the actual offset in the bitfile.

Finally, the newly created offspring is assigned to the population of the next generation, and the OGA repeats the same steps until an adequate solution is found, which in our experiments was defined as achieving no discrepancy between the evolved individual and the fault-free one over a predefined window of readings.

3.3. Communication Protocol. The communication protocol consists of two components. The first is the hardware component that resides on the FPGA board and includes the standard JTAG interface serial port and the GNAT platform [21], which is configured on the device to support input and output operations with the AE. The second component is the software, which runs on a host PC that is connected to the FPGA. The messages are 16 bits in width; therefore, the communication link can theoretically handle up to 300,000 messages per second with a Xilinx Parallel Cable IV download rate of 5 Mbps [32]. SMART has a customizable message-exchange frequency that can be controlled via a settings file.

The JTAG boundary scan interface (IEEE 1149.1) is implemented on the nonreconfigurable area of the Xilinx Virtex devices. The interface offers half-duplex serial communication between the user circuit on the FPGA and the host PC. The GNAT component is implemented on the reconfigurable area of the chip to connect the JTAG boundary scan with the user circuit to provide a bidirectional communication channel. The communication protocol relies on handshaking to acknowledge received messages and to request new ones. The protocol also specifies a 16-bit packet format with 5 bits reserved for the opcode, thus supporting up to 32 message types, while the remaining fields are used for AE and FE addressing and for other purposes, such as performance readings and component status. Examples of the messages that flow from the software layer to the hardware layer are as follows:

FE_STATUS_REQUEST: request the status of a particular FE,

AE_STATUS_REQUEST: request the status of a particular AE,

PERFORMANCE_REQUEST: request the value of the Performance Counter (PM).

The respective responses of the hardware layer to these messages are as follows:

FE_STATUS_REPORT: reports FE status (such as online fault-free, online faulty, and off line),

AE_STATUS_REPORT: reports AE status (such as simplex, duplex, and triplex),

PERFORMANCE_REPORT: reports the PM value.

4. Dynamic Partial Reconfiguration

SMART relies on the repetitive reconfiguration of the FPGA to achieve active repair via scrubbing and intrinsic evolution. Dynamic PR was successfully introduced into the RARS hardware to reduce repair time. The size of the partial bitstream of the edge-detection FE is 31,350 bytes, which results in a reconfiguration time of 49 milliseconds. However, the full bitstream size for the original design without PR is 1,712,626 bytes, which corresponds to a reconfiguration time of 2.61 seconds. As a result of introducing dynamic PR into this design, the faulty FEs could be reconfigured in 1.8% of the time originally required to reconfigure the entire system. This improvement becomes extremely important during the repair process, considering that the OGA may require thousands of evaluations to evolve an adequate solution. This approach also has the added advantage of keeping the system on line during bitstream downloading.

early access partial reconfiguration (EAPR) design flow [33] was used to achieve dynamic PR capabilities. This flow requires a strict design routine that does not follow the conventional single-pass of synthesis, mapping, and PAR. Instead, it requires the design to have an explicitly modular structure such that the PR modules are singled out at the top-level module. These modules are called Partial *Reconfigurable Modules (PRMs)*, and the region of the fabric to be reconfigured is defined as a *partial reconfigurable region (PRR)*. PRMs define the functionality of each PRR. All the other logic in the design is termed *static logic*. All resources required for an FE must be confined within a PRR.

The connection between each FE and the adjacent logic was accomplished using a number of 8-bit *bus macros (BMs)*. BMs are made available by Xilinx to compensate for the old alternative of hard-wired *tristate buffers (TBUFs)*, which have been used in earlier PR flows and are known to present strict constraints on communication bandwidth due to the limited number of TBUFs available on the fabric.

The configuration frame of Xilinx Virtex 4 FPGAs is 16 CLB long and 1 CLB wide. In RARS, each FE requires 112 CLBs, which corresponds to seven logic-configuration frames. Figure 6 shows a snapshot of each of the three PRRs along with the static, top-level full design of the RARS. The placement of the PRR and the bus macro was achieved with the help of the Xilinx PlanAhead tool [33].

5. The Repair Cycle and Self-x Properties

The controlled emergence of self-x properties is what distinguishes OCs from other design paradigms [19]. Rather than providing all alternative execution paths at design time, the system is equipped with the innate capability to actuate different configurations based on run-time sensory information, making it adaptive to diverse execution scenarios.

Figure 7 shows the repair cycle that the system executes to maintain the highest possible correct throughput. The flow diagram is partitioned into three black-framed boxes to signify the observed organic self-x properties that emerge upon executing each repair stage.

The left side of the diagram shows the organic repair that is implemented on the FPGA device. The prominent observed self-x properties are self-monitoring and self-organization. The self-monitoring property is manifested by the system's self-awareness of any discrepancy that results from one or more faulty FEs through the use of different sensors, which is enhanced with self-diagnosis of the exact faulty FE through monitoring the discrepancies of the output lines. The first repair action the system takes upon detecting faults is reorganizing the components of the system to mask the fault. The self-organization property emerges through adjusting the redundancy configuration of RARS to hide the effect of hardware failures. The example in Figure 7 shows a Duplex-TMR-Duplex reorganization scenario, but

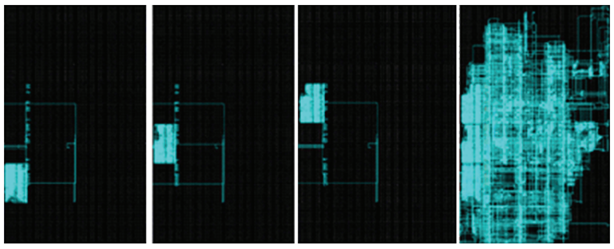
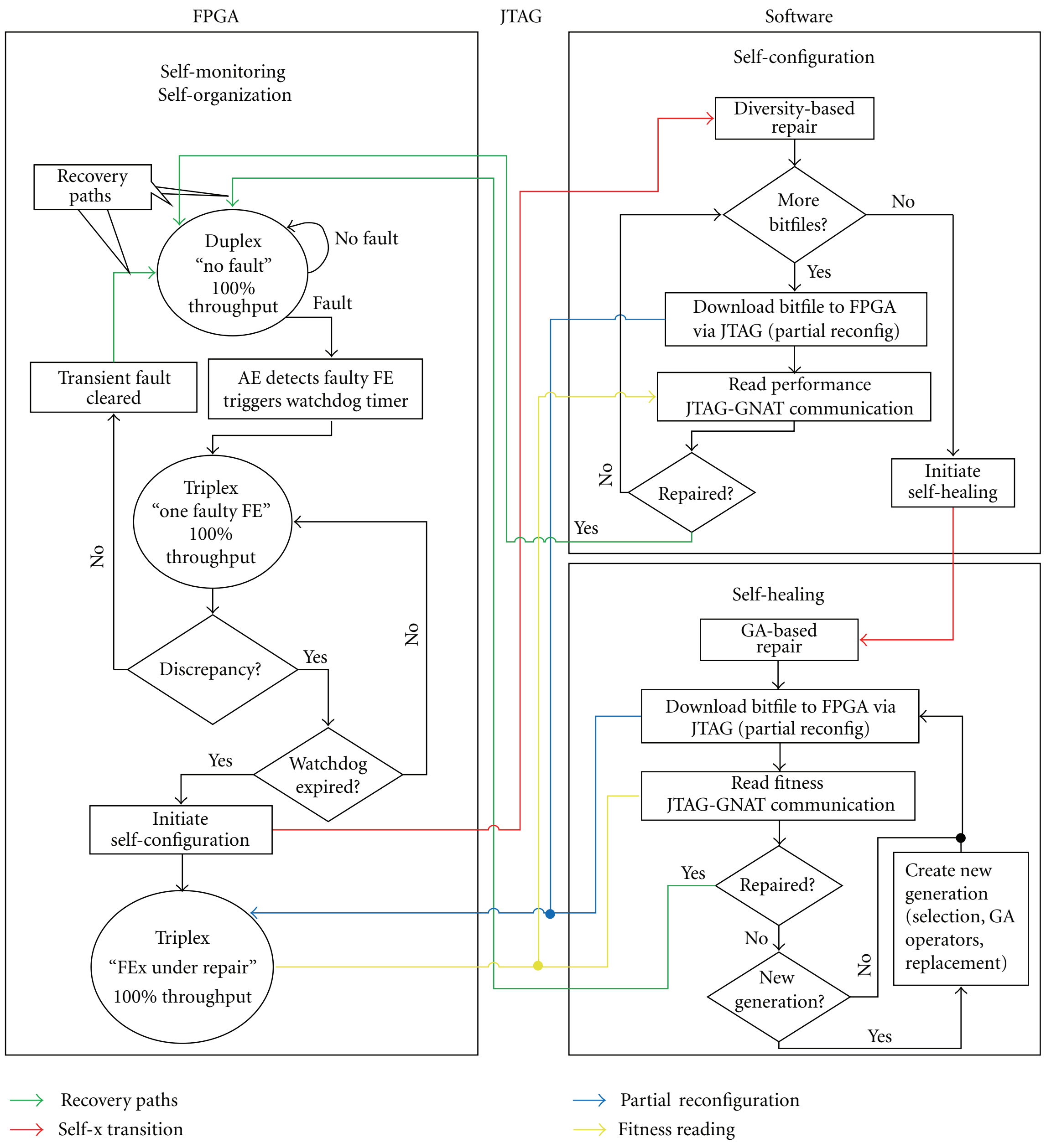


Figure 6: FE1, FE2, FE3, and the complete circuit.


FPGA
JTAG
Software
Self-monitoring
Self-organization
Recovery paths
Duplex "no fault" 100% throughput
No fault
Fault
Transient fault cleared
AE detects faulty FE triggers watchdog timer
Triplex "one faulty FE" 100% throughput
No
Discrepancy?
Yes
No
Watchdog expired?
Yes
Initiate self-configuration
Triplex "FEx under repair" 100% throughput
Self-configuration
Diversity-based repair
More bitfiles?
No
Yes
Download bitfile to FPGA via JTAG (partial reconfig)
Read performance JTAG-GNAT communication
No
Repaired?
Yes
Initiate self-healing
Self-healing
GA-based repair
Download bitfile to FPGA via JTAG (partial reconfig)
Read fitness JTAG-GNAT communication
Yes
Repaired?
No
New generation?
No
Yes
Create new generation (selection, GA operators, replacement)
Recovery paths
Self-x transition
Partial reconfiguration
Fitness reading
Figure 7: System self-x properties flow and transitions.

other reorganization sequences can be applied to meet the reliability levels stipulated by the mission requirements.

When the degree of the faults exceeds the inherent redundancy capacity of RARS, SMART triggers a different repair cycle that demonstrates another organic activity, namely, self-configuration. The self-configuration property emerges through successive lazy-scrubbing cycles, which is a process that begins by rewriting the same CBS to eliminate SEUs in the configuration logic. Then, if the fault is caused by a stuck-at hard fault in the configuration logic, scrubbing proceeds to reconfiguring the FPGA with a preseeded set of bitstreams that have different area constraints to potentially introduce an FE that excludes the faulty LUT.

Finally, if self-configuration fails to bypass the faulty element(s), the system initiates a more elaborate refurbishment cycle that relies on OGA. This evolutionary repair introduces a self-healing property at the system level that is characterized by the system's ability to actively recover from more catastrophic fault scenarios by searching for innovative solutions using evolutionary approaches. Self-healing is not limited by the degree of redundancy nor by the number of preseeded bitfiles, which makes it a compelling option for complex fault scenarios. However, SMART makes OGA the last resort in the repair sequence due to its long repair time.

A key consideration in our technique is that reconfiguration adds minimal additional component to functional critical path. The design attempts to promote the fact that, if faults occur outside the FEs logic, only the recovery is impacted, not the FEs functionality. Therefore, we can apply the RARS concept recursively if needed to provide coverage for faults in the AE. Nonetheless, reconfiguration capability needs to remain intact for recovery by reconfiguration to remain viable, and also the voter logic should remain intact, as in conventional TMR approach, to guarantee that the correct vote is propagated as the functional output.

Assuming that the AE voting core is an unbreakable voting element will indeed add a single-failure point to the fault-tolerant system. However, this risk is alleviated by the fact that the core voter element of the AE has much lower area than the FEs, meaning that the probability of fault hitting the voter element is reduced accordingly. The FEs in the experimental use case of the edge detector have a total size of approximately 1800 LUTs, compared to the voter element of approximately 100 LUTs. This means that the probability that a fault happens in the voter is 5% of the probability of a fault to hit the FEs logic. We still believe that this value cannot be neglected, we plan to extend our future work to handle this case via random pairings and temporal voting techniques that we have successfully demonstrated in [34]. Moreover, the FEs are expected to considerably increase in area for real complex applications, and the voter is not expected to scale with the same degree, further reducing the chance of broken golden element compared to the functional elements.

In SMART, failures in the reconfiguration logic will only cause the loss of the software-based fault-tolerance features, that is, scrubbing and OGA. However, the inherent organic hardware of RARS will remain intact to switch among the available simplex, duplex, and triplex configurations. This graceful degradation property means that the system will become, at worst, a TMR system if the parallel/serial interface fails.

Table 3: System modules implementation.

Module	Implementation platform
Organic layer	ML402 (lower board of VSK) with Virtex-4 FPGA (XCV4SX35)
Video capturing/buffering	Video IO Daughter Card (VIODC) (Upper board of VSK) with Virtex-2 PRO FPGA (XCV2P7)
HW-SW connection	JTAG-GNAT from FPGA sideXilinx Parallel port from PC side
Communication Class	Multithreaded C++ application
HIM	C++ Message encoder/decoder
GUI monitor	Java applet
Application	Sobel edge detector (Verilog)
OGA engine	C++ based Standard GA [21]
OGA interface	C-based API (MRRA) [23]

Complete handling of failures in reconfiguration circuitries in FPGA devices is beyond the scope of this work, and we relied on proven solutions provided by Xilinx, the main manufacturer of FPGA chips, to deal with this type of faults [35]. Moreover, the same techniques used in handling faults in the datapath can be extended to the reconfiguration logic. One prominent approach in dealing with this kind of faults using redundancy can be found in [36].

Virtex-4 FPGAs are fully characterized for single-event functional interrupt (SEFI) [35], which are SEEs that cause device-wide operation interrupts such as power on reset (POR), configuration circuitry, frame address register used extensively in the reconfiguration process, and some other global signals that affect reconfiguration logic and device functionality. Xilinx states that pulsing the PROG signal will result in correcting any of the aforementioned SEFIs [35].

More catastrophic faults, such as hard faults affecting the ICAP, can be recovered using redundancy techniques presented in [36]. This technique protects the ICAP logic in a similar fashion to any other user application logic: first, by having TMR inserted in the ICAP circuit using BL-TMR tools [37] to correct faulty configuration on the fly; second, by scrubbing the ICAP interface in case an SEU is suspected in the configuration logic. These techniques can be used to prevent the parallel/serial configuration interfaces from becoming a single point of failure in SMART.

6. Experiments and Results

Using the dual-board Xilinx Video Starter Kit (VSK) FPGA board [38] and the other technologies listed in Table 3, the benefits of SMART were demonstrated quantitatively [28] using a gradient-based Sobel edge-detection algorithm [14].

6.1. Experimental Setup: Edge Detection Application. To demonstrate SMART's organic capabilities, we implemented a popular edge-detection algorithm and tested it under injected transient and permanent fault scenarios. There are various

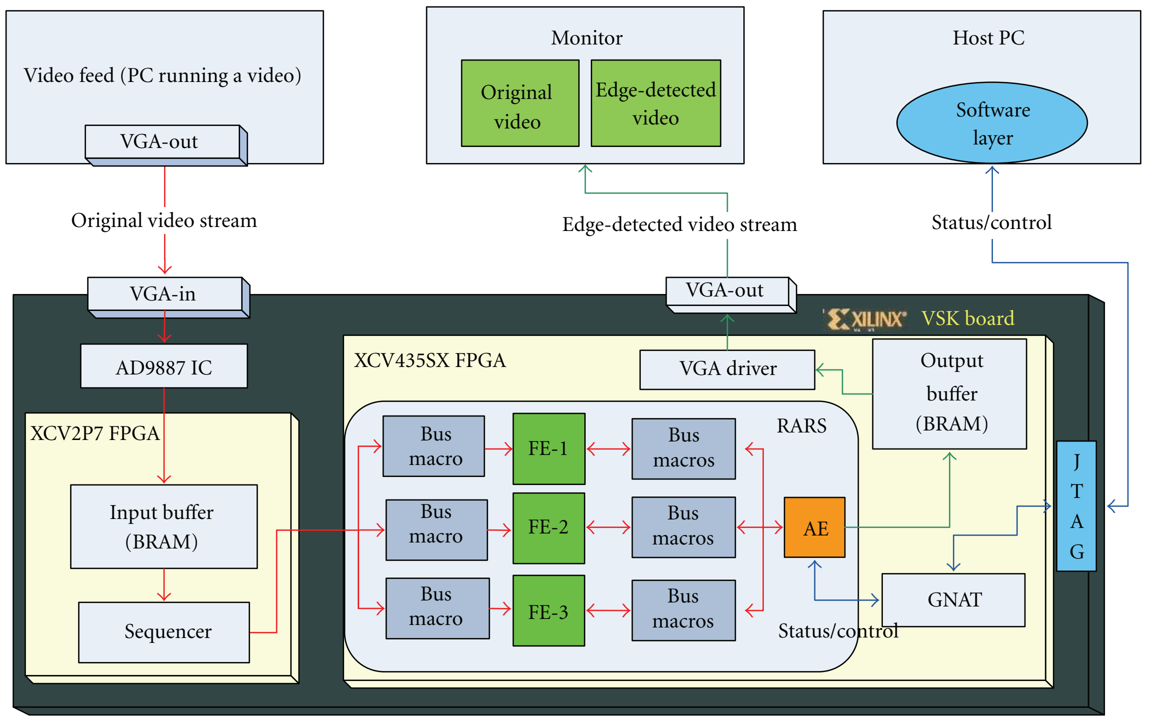


Figure 8: SMART experimental setup.

applications for edge detection because it primarily involves identifying boundaries in an image. Thus, it can be employed for object recognition and quality monitoring in industrial applications, medical imaging applications such as magnetic resonance imaging (MRI) and ultrasound imaging [39], and satellite imaging applications [40]. Numerous efforts have been made to design edge detectors using evolutionary techniques [39, 41]; we compare our GA's performance against those techniques in the experimental results section.

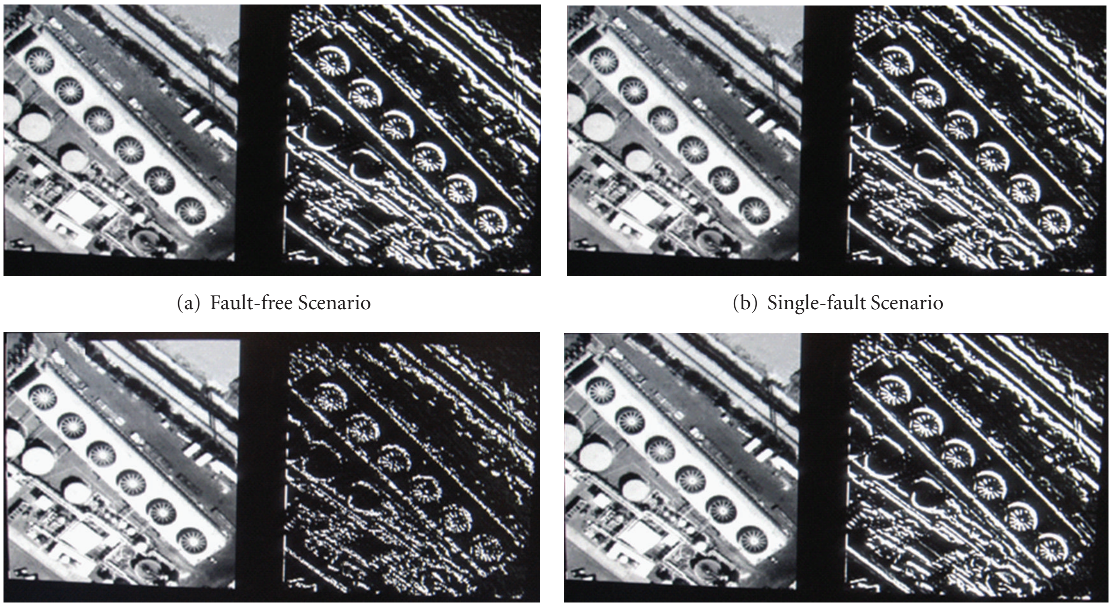
Figure 8 shows SMART application architecture where a continuous video stream provides the functional input to the circuit. The video is transmitted via either the VGA-Out or the digital video interface (DVI) ports on the host PC to the VGA-In or DVI-In, respectively, on the upper board of the Xilinx VSK, the Video IO Daughter Card (VIODC) [38]. In this system, we used the VGA ports on both ends, but nothing prevents the system from running on a DVI interface because of the versatility of the AD9887A dual interface on the VOIDC. Indeed, this IC offers both an analog and a digital receiver integrated on a single chip. The AD9887 has a parallel digital bus interface with the FPGA for video data and an I2C control bus for configuration. The captured frames are buffered into the block RAM (BRAM) of the Virtex-II Pro XCV2P7 FPGA on VIODC. The frames are continuously written on the BRAMs; if the video feed stops, then the last captured frame is used for all pixel operations until the feed is resumed.

A sequencer module handles memory scanning and synchronization and sends the pixel data through the XGI connector [38]. This connector is a 64-bit bus, called VIOBUS, which connects between the lower ML402 board and the upper VIODC. It uses a simple synchronous interface running at 100 MHz to send data and control information between the two boards.

A goal achieved in the prototype is application independence. That is, any other application can be implemented by designing new logic in the FEs and by tuning the clock-division ratio in the digital clock manager (DCM) to match the frequencies of the AE and the FEs.

Applications that are known to be more tolerant to errors than other kinds of design, such as signal processing applications, will tend to ameliorate the impact of erroneous behavior. However, the metric reported in our results is actual data path bitwise discrepancies of the output. The fitness function did not rely on any kind of pixel averaging or gradient-based operators to quantify image quality into fitness values. This discrepancy-based metric on a pixel-by-pixel basis makes this approach applicable for non-DSP applications without loss of generality.

On the ML402 motherboard, the enabled FEs in RARS process the video feed and provide the output to the AE. Based on the current configuration of the system, the AE produces the overall output and stores it in the XCV4SX35 BRAMs. In fact, the AE stores both the original and the edge-detected video stream for demonstration purposes. The BRAMs are continuously scanned by a VGA driver implemented on the same FPGA to generate the VSCAN, HSCAN, and RGB values for the VGA-out interface. The VGA-out is connected to another monitor that shows both the original and edge-detected video streams. Any error in the edge detection can be clearly spotted on this monitor, as shown in Figure 9.


(a) Fault-free Scenario (b) Single-fault Scenario

(c) Two faulty FEs scenario (d) After-repair scenario

Figure 9: Original and edge-detected images.

Table 4: DIP switch purposes.

DIP Switch	Purpose
1	AE enable to control organic capabilities
2	Stuck-at fault injected in FE1
3	Stuck-at fault injected in FE2
4	Stuck-at fault injected in FE3

Table 5: FE status LEDs.

LED 1	LED 2	FE status
OFF	OFF	Offline and faulty
OFF	ON	Offline
ON	OFF	Online and faulty
ON	ON	Online

The three FEs and the AE are connected to the host PC that runs the organic software layer. This PC is tied to a monitor that displays the real-time status of the organic layer using the GUI Java applet. The status and control signals are passed between the FEs/AE on one side and the BSCAN/JTAG on the other side. The organic layer and the FEs (i.e., the Sobel edge detector) are implemented in Verilog HDL and synthesized into FPGA bitfiles using the Xilinx ISE 9.2 software packs.

The DIP switches beneath the LCD screen on the ML402 FPGA board were used in our experiments to simulate stuck-at faults in the data path to test the ability of the RARS to switch configurations in order to mask faults immediately. One of the switches was also used to enable or disable the organic repair capabilities, as shown in Table 4. Nine LEDs were used to show the status of various modules of the design. Three of them reflected the status of the voter report shown in Table 2, whereas the other six showed the status of the FEs, with two LEDs per FE, as shown in Table 5.

It is imperative to mention that the fault simulation accomplished via the dip switches was only for the SEUs or stuck-at faults in the data path. This was done by masking the enabled dip switch logical value with one bit of the pixel input of the edge detector to affect the data signals. This kind of error should be repaired instantly by the hardware through the embedded configurations of RARS. However, to simulate stuck-at faults in the configuration logic, we had to actually alter the value of one or more of the LUT contents. We accomplished this by using the Xilinx FPGA editor tool to manually alter the content of one LUT in the NCD file in the schematic view. Both types of fault simulation were used to test the system repair cycle shown in Figure 7 and to test the intrinsic OGA repair as shown below.

The PC and JTAG prototype was only meant as a testing environment for SMART. The convenience and performance of using a PC to run the GA APIs and the communication applications have greatly reduced development time and validation effort. However, deploying the host PC with SMART will actually eliminate any benefit for such system, either from power or reliability points of view. Therefore, we believe that the system will not realize its original design goals unless it is deployed on a PowerPC processor that comes embedded within the majority of the high-end Xilinx FPGAs. To substantiate this plan, we have surveyed many of the successful PowerPC deployment efforts for fault-tolerant systems, especially ones that employ evolutionary repair techniques.

In [43], the design and implementation of an intrinsic evolution system are presented. The system relies on online evaluation of fitness, that is, using the functional input of the circuit in runtime. The GA was implemented in C (similar to

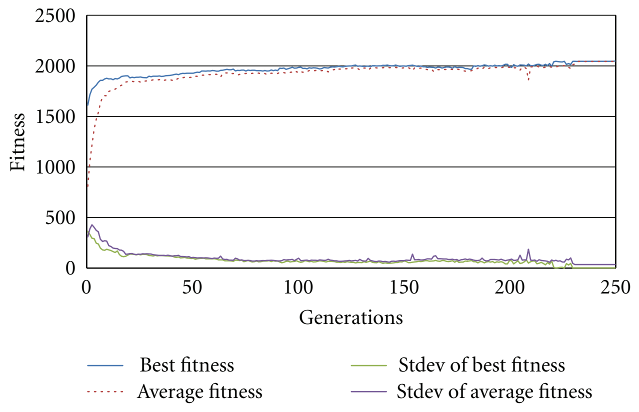


Figure 10: GA best and average fitness.

OGA in this work) and was embedded on PowerPC 405 embedded processor on a Virtex-II device. Another approach is reported in [44] where a PowerPC-based intrinsic GA and a workstation-based extrinsic GA are compared in terms of the fitness evaluation time. The intrinsic GA that evolves image recognition system was implemented on a PowerPC residing on a Virtex-II Pro FPGA, and it was shown that it achieved fitness evaluation speed comparable to software fitness evaluation that was run on a workstation operating on 30 times the frequency of the PowerPC. One might consider using the softcore that can be configured on the FPGA, like Microblaze, to achieve similar goals. However, as [45] demonstrates, softcores will consume huge number of LUTs and would consume much more power, and they are also vulnerable to the same radiation effects that can affect other logic on the board, making them far less appealing approach for fault-tolerant system implementations. Finally, the ability of IBM PowerPC to process C/C++ code [46] mitigates the risk of porting SMART into on-board implementation as all the GA and communication APIs in SMART are based on ANSI/ISO standard, the only difference being the need to interface with the ICAP rather than the parallel IV cable, which is completely supported by the PowerPC APIs [46].

6.2. Experimental Results. Figure 9(a) shows the sample input satellite image of urban buildings with industrial factory fans along with the fault-free result of real-time processing of that image using the Sobel edge-detection algorithm. Figure 9(b) depicts the scenario of single fault in the data path, which can be simulated using switches 2, 3, or 4 as defined in Table 4. Upon the detection of the discrepancy caused by the fault, the RARS switches to the TMR configuration, thereby allowing the system to maintain 100% of its fault-free throughput. Hence there is no degradation in quality compared to the fault-free scenario. Figure 9(c) depicts the impact of another stuck-at fault at a different FE, in which case system performance drops, as can be seen from the degraded edge-detected image. When the software monitoring layer initiates the refurbishment of one of the faulty FEs through PR, the system regains 100% performance, as shown in Figure 9(d). Thus, the application throughput is restored using hardware identification of resource capabilities and autonomous refurbishment.

The intrinsic bitstream evolution targeted eight LUTs in the entire FE design. These LUTs were selected after investigating the different impacts each LUT selection might have on overall system performance. Based on preliminary experiments, we were able to extract the critical LUTs [26] that are highly influential for the performance of the edge detector circuit.

The average fitness and best fitness values per generation averaged over 20 runs along with the standard deviation for both values are shown in Figure 10. The figure also shows that the standard deviation is steadily decreasing, which means that the 20 runs followed a consistent evolutionary path. The maximum fitness value in this work is 2047; this value does not actually denote the number of possible output combinations as in most conventional circuit evolution approaches. Instead, it indicates the number of discrepancies between the outputs of the evolved FE compared to another fault-free FE. To establish enough significance in the reported fitness value, the application records the number of discrepancies over a window of 65,536 evaluations, which denotes the number of pixels in one 256×256 video frame for the use case under study. Due to the message width limitation which confined the fitness value field width to 11 bits only, the hardware implemented a scaling scheme in which the actual number of evaluations of 65,536 values was scaled down by 32 to fit the field width of 11. This means that the circuit is actually evaluated for 65,536 input combinations where each 32 discrepancies are translated into 1 point on the normalized fitness scale. This technique provides wide evaluation window for the GA to span one full frame, yet avoids high transmission bandwidth for fitness reporting between the hardware and software. Another approach to expand the evaluation window while keeping the 11-bit field width is to poll the fitness values for a number of times in the OGA API and then average the readings or possibly detect and eliminate outliers; this software solution provides a way to control the number of evaluations needed to assess the evolved individual's fitness. Finally, the message width of 16 bit is just an arbitrary selection given the experimental extension of the edge detector. In real application,

Table 6: Fitness and timing information for 20 GA runs.

Run #	Final fitness		Timing information			
	Best	Average	Number of generations	Total fitness evaluation time (sec)	Total FPGA configuration time (sec)	Total genetic operators time (μsec)
1	2047	2033	147	23.69	83.50	2098.75
2	2047	2043	217	35.27	111.97	3172.50
3	2047	2006	78	12.13	35.65	1106.88
4	2047	2015	156	25.34	81.74	2421.88
5	2047	1989	99	15.96	50.09	1470.00
6	2047	2001	148	24.09	77.40	2205.00
7	2047	2005	152	25.01	79.34	2170.63
8	2047	2020	126	20.50	63.76	1835.94
9	2047	2044	252	41.27	127.01	3686.56
10	2047	2032	71	11.46	36.00	984.38
11	2047	2000	221	35.99	112.49	3093.75
12	2047	1998	162	26.27	75.82	2364.69
13	2047	2018	103	16.65	51.19	1530.00
14	2047	2044	129	21.18	64.89	1920.00
15	2047	2046	177	29.01	91.33	2585.00
16	2047	2045	161	78.80	84.85	2250.00
17	2047	2007	75	12.18	39.00	1133.13
18	2047	1993	233	38.11	117.43	3480.00
19	2047	2015	62	9.99	31.93	876.88
20	2047	2044	202	33.42	98.78	2826.56
Average		2019.90	148.55	26.82	75.71	2160.63
Standard deviation		19.80	56.73	15.40	29.00	825.48
Confidence		0.95	0.95	0.95	0.95	0.95
Alpha		0.05	0.05	0.05	0.05	0.05
95% confidence interval		(2011.2, 2028.58)	(123.69, 173.41)	(20.07, 33.57)	(63.00, 88.42)	(1798.85, 2522.4)

Table 7: Static versus partial resource usage comparison.

Approach	Full	Partial
LUTs	2785	601
Slices	1734	368
IOB	70	0
BUFGs	11	0
RAMB16s	98	0
DCM	1	0
BSCAN_	1	0

Table 8: Virtex-2 versus Virtex-4 configuration comparison.

Approach	Virtex-2 [21]	Virtex-4 Full	Virtex-4 Partial
Bitstream size	548 KB	1.633 MB	30.61 KB
JTAG cable	parallel cable III300 Kbps	parallel cable IV5 Mbps	parallel cable IV5 Mbps
Configuration time (msec)	22000	2613	49

Table 9: GA Parameters used in experiments.

Parameter	Value
Population size	10
Mutation rate	0.3
Elitism size	1
Crossover rate	0.8
Tournament size	2

the message width can be extended to 32 or even 64 bits, allowing for largest fitness value field and thus accommodating wider evaluation window.

Table 6 shows details of the 20 runs that are averaged in Figure 10. All runs converged to a final solution based on the GA parameters listed in Table 9. These parameters were determined using preliminary experiments with analysis of variance (ANOVA) study of the interaction effect between the parameters. We found that these parameter settings produced the best GA performance for this particular application. The average number of generations required to repair the fault for the 20 runs was 148 generations, while

Table 10: Comparison of the RARS prototype and three other edge-detection evolution techniques.

	Hollingworth et al. [41]	Gudmundsson et al. [39]	Ross et al. [42]	SMART
Application	Generic images (fairly simple)	Unfragmented, localized thin edges in medical images.	Microscopic images from mineral samples.	Generic (satellite images, uniform patterns, etc.).
Methodology	Exploit inherent parallelism in images	Split image into linked subimages. Maintain links between adjacent pixels.	Implement a training stage (requires sampling 23.6% of image), followed by genetic programming.	Evolve a subset of the edge detector (i.e., critical LUTs) to recover from faults.
Fitness evaluation	Software model	Software model	Software model	Intrinsic evolution
Evolutionary algorithm	Genetic programming.	2D genetic algorithm with problem-specific operators.	Genetic programming training (∼25%) and evolution (∼75%).	Genetic algorithm.
Genetic String coding	Four node functions (i.e., and, or, not, and xor) and eight terminal values for pixels around the evolved pixel.	Edge map. Image pixels are masked with corresponding values in pixel map (i.e., 0: no edge, 1: edge).	High-level functions (i.e., avg, min, max, and stdev). Terminal pixels and high-level ephemerals (i.e., gradient and intensity).	Direct bitstream evolution. The solution coding is the actual bitfile.
Fitness function	Pratt figure of merit (PFM) relative to fault-free Sobel edge detector	Highly complex cost function based on five cost factors.	Biased random sampling fitness evaluation for training. Program fitness is similar to PFM.	Model-free, triplex discrepancy-based function. No application-specific a priori knowledge needed.
Evolution speed	Partial solution in 2,333 generations after 24 hours of evolution time.	2,300 generations used for ring imaging; 300 generations used for thin, well-localized edges.	75 generations, with 25% of images used for training. Very large population size of 2,000.	148 generations, with low population size of 10. Evolved 8 critical LUTs.
Best fitness	Not reported	0.85 PFM with scaling factor of 0.01.	0.590 for Image 1;0.633 for Image 2.	100% as compared to output from fault-free Sobel edge detector.

the 95% confidence interval was between 123 and 173 generations. Thus, 95% of the time, a repair happens in less than 173 generations.

The runs produced very small deviation in the average fitness of the population; this is partly due to the small population size. The table also shows the timing information for fitness evaluation, PR time, and genetic operator overhead. Although we used PR, the configuration time was still the dominant factor in repair time. For example, the first run required 83.5 seconds of reconfiguration time for a total of 147 generations, which means that each generation required $83.5 \div 147 = 0.568$ seconds to configure a population of 10 individuals, resulting in 56.8 msec per bitfile. This is close to the theoretical value obtained by dividing the bitstream size by the cable speed ($30.61\,\text{KB} \div 5\,\text{Mbps} = 49\,\text{msec}$). This value accounts only for CBS transmission time. In reality, there is a 95% probability that the download time will take 64 to 88 msec.

Tables 7 and 8 rely on the results of our previous work on Virtex-2 [21] to compare resource usage, utilization, and configuration details between the Virtex-2 platform and the Virtex-4 SMART platform for the full and partial bitstream cases. These tables show the advantages of using PR in terms of bitstream size, download time, and resource requirements.

Table 10 compares the edge-detection evolutionary approach that was implemented in this work and the other three approaches found in the literature [39, 41, 42], conveying the merits of intrinsic evolution. In addition, the model-free fitness function provides an application-independent approach compared to the complex fitness functions adopted by the others. Finally, OGA performance and results show that the PR paired with direct bitstream evolution yielded evolutionary repair in 148 generations on average.

6.3. The Relationship between RARS and the OGA. RARS is the hardware organic component of SMART. Its primary purpose is masking transient faults resulting from SEU in the user logic until the affected user register is rewritten with new value from the datapath, thus eliminating the SEU impact. In addition, RARS helps maintaining correct functional output even in the case of soft faults in the configuration logic, until the scrubber redownloads the CBS and corrects the upset. However, in the cases of hard faults, RARS cease to be efficient as it does not have the mean to find alternative paths at the LUT granularity of the FEs. Here comes the role of the OGA, which will be invoked by SMART's controller to realize solutions even in the case of hard faults. Still, RARS plays significant role in hard-fault repair by interacting with the OGA in the following ways.

(1) As the OGA is a guided heuristic search method that requires evaluating many individuals until a good solution is found, and because the OGA performs online fitness assessment, meaning that the evaluated individual is configured on the circuit and is evaluated using the runtime functional inputs that drive the application, RARS conceals the effect of evaluating suboptimal individuals by switching to TMR mode so that the—sometimes—erroneous outputs of the evaluated individual are overruled by other fault-free FEs, just as if the evolved FE is affected by a transient fault. This will give the OGA enough time to evolve optimal individual without affecting the functional operation of the circuit.

(2) The OGA relies on the self-monitoring capabilities or RARS, which evaluates the evolved FE and presents its fitness value to the OGA engine. The OGA by itself cannot assess the fitness of the intrinsically evolved individual and thus needs to interact with RARS.

To demonstrate this behavior, we applied a sequence of injected faults on RARS and monitored the performance of the application. Then, we superimposed OGA repair experiments taken under the same conditions to create a holistic experiment that exploits the two pieces together. The precondition for this sequence of events is that hard-fault MTTR should be greater than the MTBF; this condition is almost always realized in space missions due to low MTTF in radiation-hardened FPGA devices that employ epitaxial CMOS process technology to lessen the impact of energetic particles hitting the silicon. As seen in Figure 11, the number of faulty FEs in RARS goes from 1 to 2 to 3 by time, as there is no mechanism to repair hard faults. On the other hand, hard faults in FEs are corrected as they occur to maintain a number of faulty FEs less than or equal to 1.

With the help of RARS, this guarantees a steady 100% overall performance of the application, even though the faulty FEs are being evaluated on line with performance levels down to 15% at some point of time. The non-OGA mode will eventually suffer degraded operation when there are two faulty Fes; this is because the voting mechanism cannot guarantee correct vote. With 3 faulty FEs, the overall performance of RARS gets closer to 50% as the three FEs can generate faulty outputs. The voter hits this performance level due to compensating fault scenarios in which the FEs do not fail in the same way and thus can still vote for the correct output in about 50% of the cases.

6.4. Power Savings. A conventional TMR configuration employs three times the required user logic plus a voter to ensure that the system can tolerate faults in finite portions of the mission. In the current state-of-the-art technology, in which a single FPGA chip can contain millions of gates at a low price, power consumption continues to be the prevailing concern of hardware-redundancy fault-tolerance approaches, especially in missions that operate under stringent power limitations.

However, RARS operates under the minimal level of redundancy necessary to satisfy mission requirements and only enables redundant parts as they are needed. This design results in power savings under the no-fault scenario, which usually constitutes the majority of an application's lifetime. To analyze the power savings of RARS over TMR, let the quantities of interest be denoted as follows:

P_{FE}: dynamic power consumption of one FE,

P_{AE}: dynamic power consumption of one AE (without the voter component),

P_{V}: dynamic power consumption of the voter,

P_{TMR}: dynamic power consumption of conventional TMR,

P_{RARS}: dynamic power consumption of RARS,

P_S: power savings by using RARS over conventional TMR

$$P_S = \frac{P_{\mathrm{TMR}} - P_{\mathrm{RARS}}}{P_{\mathrm{TMR}}}. \tag{2}$$

The power consumed by conventional TMR is three times the FE power plus the power of the voter:

$$P_{\mathrm{TMR}} = 3 \times P_{\mathrm{FE}} + P_{\mathrm{V}}. \tag{3}$$

In RARS, power consumption is divided into two components, namely, power consumption when the system is fault-free and power consumption when the system is under repair:

P_R: RARS power consumption when the system is fault-free,

P_{1-R}: RARS power consumption when the system is under repair,

R: reliability of the system, defined as having no single or multiple faults in any of the FEs

$$P_{\mathrm{RARS}} = R \times P_R + (1 - R) \times P_{1-R}. \tag{4}$$

Assume a simple scenario for RARS such that it runs in duplex mode until the DS detects a disagreement between the outputs of the two enabled FEs, at which point RARS enables the voter and the stand by FE to implement the TMR configuration. Upon removal of the fault, RARS returns to duplex mode. Thus, the system has duplex power consumption under normal operating conditions (R) and TMR power consumption under single or multiple faults mode $(1 - R)$:

$$\begin{aligned} P_R &= 2 \times P_{\mathrm{FE}} + P_{\mathrm{AE}}, \\ P_{1-R} &= 3 \times P_{\mathrm{FE}} + P_{\mathrm{AE}} + P_{\mathrm{V}}. \end{aligned} \tag{5}$$

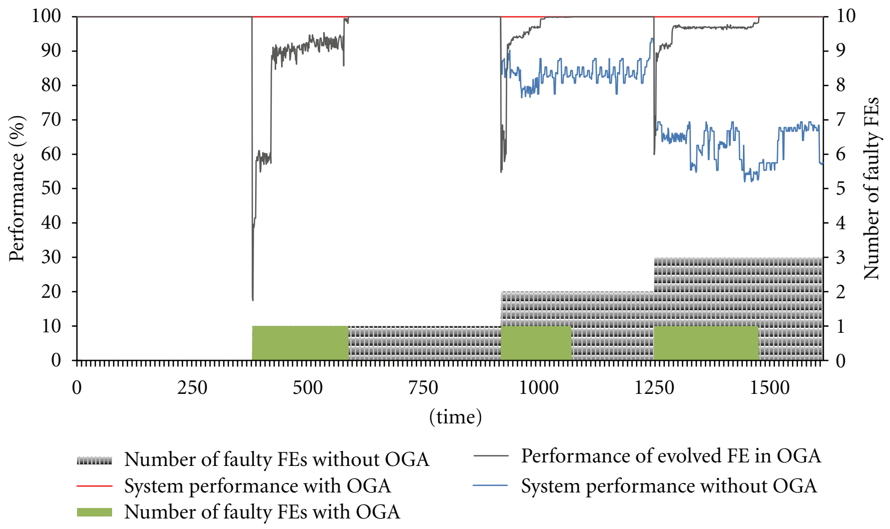


FIGURE 11: Holistic experiment to demonstrate the interaction between RARS and OGA.

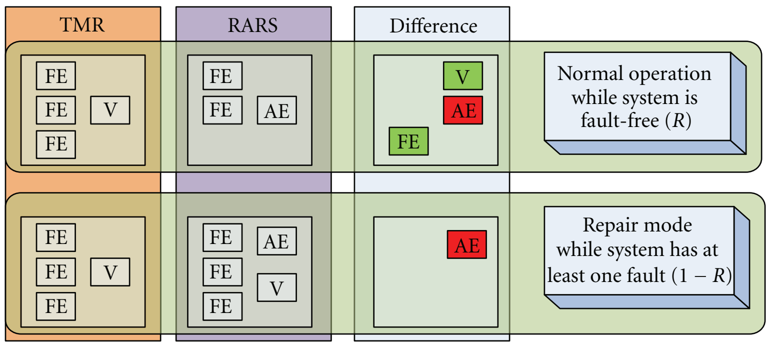

TMR
RARS
Difference
FE
FE
FE
V
FE
FE
AE
V
AE
FE
Normal operation while system is fault-free (R)
FE
FE
FE
V
FE
FE
FE
AE
V
AE
Repair mode while system has at least one fault (1 − R)

FIGURE 12: Component difference between RARS and TMR.

Substituting (3) and (4) into (2) results in

$$P_S = \frac{3 \times P_{FE} + P_V - [R \times P_R + (1 - R) \times P_{1-R}]}{P_{TMR}}. \quad (6)$$

Substituting (5) into (6) results in

$$P_S = \frac{R(P_{FE} + P_V) - P_{AE}}{P_{TMR}}. \quad (7)$$

Equation (7) represents the power savings of using RARS over conventional TMR. Thus, increasing system reliability (R) results in increased power gain for RARS because the system can benefit from prolonged operation under the low-power profile. The same equation can be inferred by comparing the difference between TMR and RARS under two operation modes, namely, normal operation (R) and repair operation ($1 - R$).

Figure 12 can be used to substitute ($P_{TMR} - P_{RARS}$) into (2) by accounting for the difference between the two configurations:

$$P_{TMR} - P_{RARS} = R \times (P_{FE} + P_V - P_{AE}) + (1 - R) \times (-P_{AE}). \quad (8)$$

Substituting (8) into (2) produces the same P_S equation in (7).

To experimentally qualify the analytical results, we employed the Xilinx XPower analyzer tool (XPA) [47] to measure the dynamic power consumption for the different system components. The XPA is part of the Xilinx ISE design suite and provides a way to analyze the power profile of post-PAR designs, which has an advantage over the alternative tool, Xilinx power estimators (XPEs) [47], which relies only on mapping reports and ignores the details of placement and routing in estimating power consumption. Table 11 shows the XPA settings that we used, and Table 12 shows the measured dynamic power for the various components.

Substituting the values of Table 13 into (8), RARS obtains an advantage over TMR in power consumption at moderately low reliability ratings. For instance, using (3), and (5) $P_{TMR} = 9.91$ mWatts, $P_R = 6.98$ mWatts, and $P_{1-R} = 10.71$ mWatts. Table 13 lists P_S for selected reliability values.

As shown in Figure 13, plotting the P_S function versus reliability produces linear improvement in RARS power consumption over TMR as reliability values increase. The cutoff point when RARS becomes advantageous in terms of power saving is $R = 0.214$, which is a rare, very low reliability

Table 11: Xilinx XPower analyzer settings.

Setting	Value
Airflow (LFM)	250
Ambient temp.	25
Toggle rate (%)	12.5
Fanout for clock	Actual
Fanout for logic	Actual
Heat sink	No

Table 12: XPower analyzer results (power in mWatts).

Component	LUTs	User FF	Signals power	Logic power	Total power
FE	526	138	1.21	1.88	3.09
AE	136	78	0.25	0.55	0.8
Voter	107	0	0.12	0.52	0.64

Table 13: XPA power savings results.

Reliability	P_S
1%	−7.07%
21%	−0.17%
22%	0.21%
50%	10.75%
70%	18.27%
90%	25.8%
97%	28.44%
99%	29.19%
100%	29.57%

value in real-life mission-critical systems because it implies that the mission is under repair 80% of the time. RARS offers 20% power savings at R values as low as 0.75, and the gain approaches 30% for high-reliability systems ($R > 0.95$) while retaining similar fault-protection capabilities to those associated with TMR.

Additional savings can be achieved if the application can accommodate simplex configurations for part of the mission, especially if some data-encoding scheme can accomplish a sufficient level of fault detection. Also, the DS can be further power-gated in the TMR configuration of RARS to save more power while the system is under repair. Finally, the implemented FE was for demonstration purposes and thus consumed limited power compared to real-life complex applications. The AE power budget will not scale proportionally with the FEs when increasing application complexity. This means that P_S should improve even further when using RARS in more complex applications.

Neglecting the reconfiguration process power consumption can only be acceptable if the configuration time is very low compared to the application running time, which can be achieved by two ways.

First, by reducing the bitstream size through the use of PR rather than full configuration approach. As we have demonstrated in the experimental section, the bitstream size was reduced using the PR flow to 30.61 KB compared to 1.633 MB for the full static bitstream. This reduction in CBS size led to decreasing reconfiguration time to 1.8% of the original value, which should translate into comparable power saving during the reconfiguration process.

Second, the configuration time can be vastly reduced by relying on the much faster internal configuration port (ICAP) instead of external configuration ports such as the JTAG. As mentioned previously, and in spite of the usage of the parallel Cable IV in the experimental setup, the intended deployment platform which will utilize the PowerPC processor will make use of the ICAP for all reconfigurations. The ICAP can reach download speed of up to 400 MB/Sec compared to the 5 MB/Sec for the parallel Cable IV that we used in experimental setup. The problem that faces most designers is that this speed is bounded by the limiting factor of fetching the CBS from the configuration memory into ICAP with the same rate. Thus, ICAP is able to support the maximum throughput 400 MB/Sec, but the bottleneck becomes how fast the application can fetch the configuration data from memory.

Several efforts in the literature have implemented CBS fetching mechanisms to match the speed of the ICAP. In [48], an implementation of BRAM next to the ICAP along with a finite state machine (FSM) to drive the memory load operations into the ICAP is presented. The resulting system was able to write 4-byte words to the ICAP at a frequency of 100 MHz, matching the maximum throughput made available by ICAP. In [4], the lightweight hardware artNOC-ICAP interface is developed to support fast readback-modify-writeback (RMW) mechanism that achieved 40 us configuration time per frame, again matching the maximum speed of the ICAP. Another successful approach to match the ICAP speed is presented in [49], based on direct memory Access (DMA) aided by master burst and BRAM caching technique. Another extensive effort is demonstrated in [50] where the JTAG dynamic power consumption is measured via a digital oscilloscope from a Spartan III FPGA that does not have an ICAP interface. The reconfiguration time for a PR bitfile of 21 KB was 34 ms, utilizing ICAP instead with a performance of 66 MB/Sec on a Virtex II device would reduce the configuration time to 0.32 ms, and this 99% reduction in configuration time would again yield considerable reduction in reconfiguration power.

7. Conclusion

A dual-tiered approach can adapt to run-time failures based on alternative FPGA configurations. This allows the use of a continuum of power utilization versus reliability. The organic hardware approach provides decentralized awareness and control by means of distributed RARS modules across the hardware fabric. The supervisory software layer provides the ability to assimilate hardware sensory information while providing vital high-level decision-making.

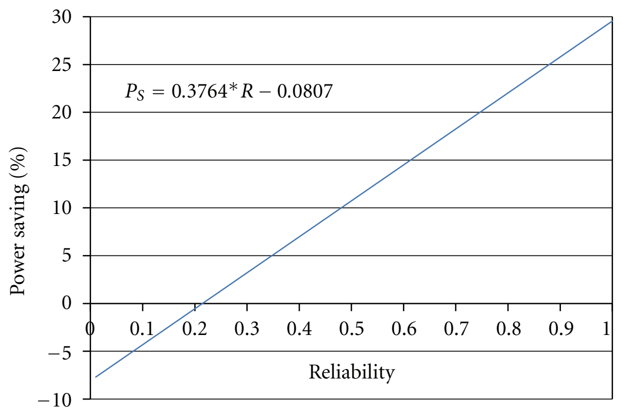


FIGURE 13: RARS power savings over TMR.

RARS avoids the dilemma of choosing a fixed degree of redundancy by deferring commitment to a particular fault-handling configuration until run-time. This approach, utilizing the reconfigurability of SRAM-based FPGAs, demonstrates an effective use of resources depending on current mission conditions. TMR consumes three times the default resources required to survive during the short periods of time when faults hit the circuit. RARS, in contrast, adapts to the various requirements at different stages of the mission by enabling just the right number of spares. Unnecessary spares can be completely disabled or even replaced by other circuits. In the age of power-aware devices, in which cooling and battery life are as crucial as performance, RARS is able to save up to 30% of the power used by TMR while still providing protection against transient and permanent faults.

PR has made it possible to keep the system on line while being repaired. Moreover, it has allowed the implementation of active fault-handling techniques at the software layer, such as lazy scrubbing and intrinsic fitness evaluation.

The software layer relies on a JTAG interface to communicate with the FPGA and to download partial bitfiles. This layer facilitates experiments with evolutionary repair in which recovery ability is not limited by the number of spares available. OGA, unlike other conventional GAs, supports features matched to OC requirements. The model-free fitness function guarantees the portability and scalability of the GA to fit any application domain. Direct CBS evolution reduces the mapping time of the genetic material onto physical individuals, thereby boosting the performance of the GA. Finally, intrinsic evolution improves the accuracy of the GA because it allows evolution of the system on the actual hardware rather than on a software model.

Now that we have successfully demonstrated the ability to implement power-conservative evolutionary-based OC on reconfigurable devices, in future research we aim to incorporate the random pairings and temporal voting technique that we presented in [34] to eliminate the dependency on the AE as a golden element in RARS. Moreover, we plan to improve the performance of the OGA to achieve further reduced repair times that fit the critical nature of the missions employing such fault-tolerant systems. We intend to investigate the prospect of parallelizing the OGA under the Island-GA model to leverage the distributed nature of OC systems on reconfigurable devices.

Acknowledgment

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) under Contract no. W31P4Q-08-C-0168.

References

[1] H. Schmeck, “Organic computing-a new vision for distributed embedded systems,” in *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC ’05)*, pp. 201–203, Washington, DC, USA, May 2005.

[2] C. Müller-Schloer, “Organic computing: on the feasibility of controlled emergence,” in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 2–5, Stockholm, Sweden, September 2004.

[3] G. Lipsa, A. Herkersdorf, W. Rosenstiel, O. Bringmann, and W. Stechele, “Towards a framework and a design methodology for autonomic SoC,” in *2nd International Conference on Autonomic Computing, (ICAC ’05)*, pp. 391–392, Washington, DC, USA, June 2005.

[4] S. Christian, H. Bastian, and J. Becker, “An interface for a decentralized 2d reconfiguration on xilinx virtex-FPGAs for organic computing,” *International Journal of Reconfigurable Computing*, vol. 2009, 2009.

[5] J. Haase, A. Hofmann, and K. Waldschmidt, “A self distributing virtual machine for adaptive multicore environments,” *International Journal of Parallel Programming*, vol. 38, no. 1, pp. 19–37, 2010.

[6] M. Parris, C. Sharma, and R. Demara, “Progress in autonomous fault recovery of field programmable gate arrays,” Accepted to, *ACM Computing Surveys*.

[7] G. Asadi and M. B. Tahoori, “Soft error rate estimation and mitigation for SRAM-based FPGAs,” in *Proceedings of the 13th ACM/SIGDA ACM International Symposium on Field*

Programmable Gate Arrays, (FPGA '05), pp. 149–160, New york, NY, USA, February 2005.

[8] F. Lima, L. Cairo, and R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs," in *Proceedings of the 40th Design Automation Conference*, pp. 650–655, Anaheim, Calif, USA, June 2003.

[9] R. DeMara, J. Lee, R. Al-Haddad et al., "Invited paper: dynamic partial reconfiguration approach to the design of sustainable edge detectors," in *Proceedings of the Engineering of Reconfigurable Systems and Algorithms (ERSA '10)*, p. 11, Las Vegas, Nev, USA, 2010.

[10] S. Mitra, N. R. Saxena, and E. J. McCluskey, "A design diversity metric and reliability analysis for redundant systems," in *Proceedings of the International Test Conference (ITC '99)*, pp. 662–671, September 1999.

[11] M. Garvie and A. Thompson, "Scrubbing away transients and Jiggling around the permanent: long survival of FPGA systems through evolutionary self-repair," in *Proceedings of the 10th IEEE International On-Line Testing Symposium, (IOLTS '04)*, pp. 155–160, July 2004.

[12] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *Proceedings of the 11th International Workshop, Field-Programmable Logic and Applications and Lecture Notes in Computer Science*, pp. 99–104, August 2009.

[13] S. Vigander, *Evolutionary Fault Repair of Electronics in Space Applications*, Norwegian University Science and Technology, Trondheim, Norway, 2001.

[14] I. Sobel, *Camera models and machine perception*, Ph.D. thesis, Department of Computer Science, Stanford University, 1970.

[15] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 212–221, 1998.

[16] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriya, and V. Verma, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications," in *Proceedings of the International Test Conference (ITC '99)*, pp. 973–982, September 1999.

[17] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmabletransistor arrays," *IEEE Transactions on Reliability*, vol. 49, pp. 305–316, 2000.

[18] A. Bouajila, A. Zeppenfeld, W. Stechele et al., "Organic computing at the system on chip level," in *Proceedings of the IFIP International Conference on Very Large Scale Integration and System-on-Chip, (VLSI-SoIC '06)*, pp. 338–341, October 2006.

[19] J. Branke, M. Mnif, C. Müller-Schloer et al., "Organic Computing-addressing complexity by controlled self-organization," in *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, (ISoLA '06)*, pp. 185–191, Paphos, Cyprus, November 2006.

[20] J. Becker, K. Brändle, U. Brinkschulte et al., "Digital on-demand computing organism for real-time systems," in *Proceedings of the 19th International Conference on Architecture of Computing Systems, (ARCS'06)*, pp. 230–245, Frankfurt am Main, Germany, 2006.

[21] R. Oreifej, R. Al-Haddad, H. Tan, and R. DeMara, "Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II Pro devices," in *Proceedings of the International Conference on Field Programmable Logic and Applications, (FPL '07)*, pp. 299–304, August 2007.

[22] R. F. DeMara and K. Zhang, "Autonomous FPGA fault handling through competitive runtime reconfiguration," in *Proceedings of the ASA/DoD Conference on Evolvable Hardware, (EH '05)*, pp. 109–116, Washington, DC, USA, July 2005.

[23] H. Tan and R. DeMara, "A multilayer framework supporting autonomous run-time partial reconfiguration," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 5, pp. 504–516, 2008.

[24] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, pp. 200–209, 1962.

[25] S. Mitra and E. J. McCluskey, "Which concurrent error detection scheme to choose?" in *Proceedings of the International Test Conference*, pp. 985–994, Atlantic City, NJ, USA, October 2000.

[26] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, 2008.

[27] S. Y. Yu and E. J. McCluskey, "Permanent fault repair for FPGAs with limited redundant area," in *Proceedings of the 16th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, (DFT '01)*, pp. 125–133, October 2001.

[28] R. Al-Haddad, "RARS in action," 2010.

[29] Xilinx, ISE In-Depth Tutorial (V 9.1), 2007.

[30] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[31] G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of virtex devices through internet reconfigurable logic," in *Proceedings of the 3rd International Conference on Evolvable Systems*, 2000.

[32] Xilinx, Xilinx Parallel Cable IV, Product Specification DS097 (v2.5), 2008.

[33] Xilinx, Partial Reconfiguration User Guide, UG702 (v 12.1), 2010.

[34] R. F. DeMara and C. A. Sharma, "Self-checking fault detection using discrepancy mirrors," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA '05)*, pp. 311–317, Las Vegas, Nev, USA, June 2005.

[35] C. Carmichael and C. W. Tseng, "Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory," *Xilinx Application Note (XAPP197)*, 2009.

[36] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Proceedings of the IEEE Aerospace Conference, (AC '08)*, pp. 1–10, Big Sky, Mont, USA, March 2008.

[37] H. Quinn, P. Graham, K. Morgan, J. Krone, M. Caffrey, and M. Wirthlin, "An introduction to radiation-induced failure modes and related mitigation methods for Xilinx SRAM FPGAs," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, (ERSA '08)*, pp. 139–145, July 2008.

[38] Xilinx, Video Starter Kit User Guide UG217 (v1.5), 2006.

[39] M. Gudmundsson, E. A. El-Kwae, and M. R. Kabuka, "Edge detection in medical images using a genetic algorithm," *IEEE Transactions on Medical Imaging*, vol. 17, no. 3, pp. 469–474, 1998.

[40] J. F. Cayula and P. Cornillon, "Edge detection algorithm for SST images," *Journal of Atmospheric and Oceanic Technology*, vol. 9, no. 1, pp. 67–80, 1992.

[41] G. S. Hollingworth, S. L. Smith, and A. M. Tyrrell, "Design of highly parallel edge detection nodes using evolutionary techniques," in *Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing*, 1999.

[42] B. J. Ross, F. Fueten, and Y. Y. Dmytro, "Edge detection of petrographic images using genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 658–665, San Francisco, Calif, USA, 2000.

[43] S. Merchant, G. Peterson, S. Park, and S. Kong, "Intrinsic embedded hardware evolution of block-based neural networks," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3129–3136, Vancouver, BC, Canada, 2006.

[44] K. Glette, J. Torresen, and M. Yasunaga, "Online evolution for a high-speed image recognition system implemented on a Virtex-II Pro FPGA," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems, (AHS '07)*, pp. 463–470, Edinburgh, Scotland, UK, 2007.

[45] A. Telikepalli, "Power vs. performance: the 90 nm inflection point," Xilinx White Paper 223, 2005.

[46] E. J. McDonald, "Runtime FPGA partial reconfiguration," *IEEE Aerospace and Electronic Systems Magazine*, vol. 23, no. 7, pp. 10–15, 2008.

[47] Xilinx, Xilinx Power Tools Tutorial UG733 (v1.0), 2010.

[48] S. Liu, R. N. Pittman, and A. Forin, "Energy reduction with run-time partial reconfiguration," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, Monterey, CA, USA, February 2010.

[49] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications, (FPL '09)*, pp. 498–502, September 2009.

[50] K. Paulsson, M. Hübner, S. Bayar, and J. Becker, *Exploitation of run-time partial reconfiguration for dynamic power management in Xilinx spartan III-based systems*, ReCoSoc2007, Montpellier, France, 2007.